



**A MULTI-VEHICLE COOPERATIVE
LOCALIZATION APPROACH FOR AN
AUTONOMY FRAMEWORK**

THESIS

Edwin A. Mora, Second Lieutenant, USAF
AFIT/ENG/19M

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT/ENG/19M

A MULTI-VEHICLE COOPERATIVE LOCALIZATION APPROACH FOR AN
AUTONOMY FRAMEWORK

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Electrical Engineering

Edwin A. Mora, B.S.E.E.

Second Lieutenant, USAF

March 21, 2019

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT/ENG/19M

A MULTI-VEHICLE COOPERATIVE LOCALIZATION APPROACH FOR AN
AUTONOMY FRAMEWORK

THESIS

Edwin A. Mora, B.S.E.E.
Second Lieutenant, USAF

Committee Membership:

Dr. Robert C. Leishman
Chair

Dr. John Raquet
Member

Dr. David R. Jacques
Member

Abstract

Offensive techniques produced by technological advancement present opportunities for adversaries to threaten the operational advantages of our joint and allied forces. Combating these new methodologies requires continuous and rapid development towards our own set of “game-changing” technologies. Through focused development of unmanned systems and autonomy, the Air Force can strive to maintain its technological superiority. Furthermore, creating a robust framework capable of testing and evaluating the principles that define autonomy allows for the exploration of future capabilities. This research presents development towards a hybrid reactive/deliberative architecture that will allow for the testing of the principles of task, cognitive, and peer flexibility. Specifically, this work explores peer flexibility in multi-robot systems to solve a localization problem using the Hybrid Architecture for Multiple Robots (HAMR) as a basis for the framework. To achieve this task a combination of vehicle perception and navigation tools formulate inferences on an operating environment. These inferences are then used for the construction of Factor Graphs upon which the core algorithm for localization implements iSAM2, a high performing incremental matrix factorization method. A key component for individual vehicle control within the framework is the Unified Behavior Framework (UBF), a behavior-based control architecture which uses modular arbitration techniques to generate actions that enable actuator control. Additionally, compartmentalization of a World Model is explored through the use of containers to minimize communication overhead and streamline state information. The design for this platform takes on a polymorphic approach for modularity and robustness enabling future development.

Acknowledgements

First and foremost I would like to thank my advisor for his patience and support throughout this experience. Without his care and encouragement I would not have made it to this point. I would also like to thank AFIT and the Air Force for this wonderful opportunity to pursue my Master's degree. Le doy gracias a mi familia por su apoyo y amor incondicional. To my friends that were there for me during the tough times, I want you to know that your existence in my life is greatly appreciated and I cherish you. Lastly, I would like to offer my gratitude to those individuals that challenged me during this experience. My personal growth as a person would not have been possible without all of you.

Edwin A. Mora

Table of Contents

	Page
Abstract	iv
Acknowledgements	v
List of Figures	viii
List of Tables	ix
I. Introduction	1
1.1 Document Overview	1
1.2 Research Motivation	2
1.3 Research Objectives	3
1.4 Assumptions and Constraints	4
II. Background	6
2.1 Robotic Frameworks	6
2.1.1 Deliberative Architectures	7
2.1.2 Reactive Architectures	8
2.1.3 Early Hybrid Deliberative/Reactive Architectures	11
2.1.4 Centralized and Distributed Architectures	13
2.1.5 Arbitration Techniques	14
2.2 Middleware	16
2.2.1 Robotic Operating System	16
2.2.2 Google Protocol-Buffers	17
2.2.3 Lightweight Communication and Marshalling	17
2.3 Cooperative Localization	20
2.3.1 Bayesian Filters	20
2.3.2 Factor Graphs	21
III. Methodology	24
3.1 HAMR	25
3.1.1 Deliberator	25
3.1.2 Controller	26
3.1.3 Sequencer	29
3.1.4 Coordinator	31
3.2 Objective	33
3.2.1 Localization Algorithm	35
3.2.2 Assumptions	37
3.3 Limitations	40
3.4 Summary	41

	Page
IV. Results	42
4.1 Component Communication	42
4.2 Deliberator	43
4.3 Sequencer	44
4.4 Coordinator	45
4.5 Controller	46
4.5.1 UBF	46
4.5.2 State	46
4.5.3 Hardware Abstraction	47
4.6 Behavior Simulation	47
4.6.1 FollowPath	48
4.6.2 AvoidOthers	51
4.6.3 FollowObject	54
4.7 Cooperative Localization	57
4.7.1 Control Variables	57
4.7.2 Response Variables	57
4.7.3 Constant Factors	59
V. Conclusion	62
5.1 Summary	62
5.2 Contribution	63
5.3 Final Remarks	64
Bibliography	65

List of Figures

Figure		Page
1.	Deliberative Architecture Decomposition	7
2.	Subsumption Architecture Decomposition	9
3.	Circuit Architecture Decomposition	12
4.	Damn Architecture	14
5.	Factor Graph Nodes	21
6.	HAMR	24
7.	Coordinator Internals	32
8.	FollowPath Start of Simulation	50
9.	FollowPath End of Simulation	50
10.	AvoidOthers Start of Simulation	52
11.	AvoidOthers Collision Reaction	53
12.	AvoidOthers Collision Evasion	53
13.	AvoidOthers Return to FollowPath	54
14.	FollowObject Start of Simulation	56
15.	FollowObject End of Simulation	56
16.	LMA Computation Time	60
17.	LMA Solution Over Time	61

List of Tables

Table	Page
1. Middleware Comparison Table	19
2. State Variables	38
3. FollowPath Simulation Definitions	49
4. AvoidOthers Simulation Definitions	51
5. FollowObject Simulation Definitions	55
6. Control Variables	58
7. Response Variables	58
8. Constant Factors	59

A MULTI-VEHICLE COOPERATIVE LOCALIZATION APPROACH FOR AN AUTONOMY FRAMEWORK

I. Introduction

1.1 Document Overview

This document provides insight on the research and development made towards an autonomy framework platform. The intent behind this platform's creation is to provide researchers with a robust and capable test-bed for the development of autonomy capabilities. A Behavior-based Hybrid Deliberative/Reactive Architecture approach was the basis of this framework's initial design. Included in the framework are key components that follow a traditional *three-layer* architecture with modifications made primarily for abstracting the responsibilities of multi-vehicle management in mobile ad-hoc networks. Additionally, alternate approaches for state management are explored for the sake of streamlining a robust *World Model*. The framework developed is then tested for functionality through its implementation on a Cooperative Localization task. The algorithm for this task features a Factor Graph approach which uses *trees* to establish relationships between measurements and history by making inferences on the graph's representation of this data.

The organizational structure for this document begins with this chapter in which the motivational factors behind this research and creation for this framework are discussed. In addition, Chapter 1 also presents the objectives and desired features in the proposed framework, as well as the assumptions made. In Chapter 2, the fundamentals of pre-existing frameworks are presented with the intent of providing conceptual

insight on framework design. Additionally, background information on other integral components in this applied framework such as middleware and filter approaches for cooperative localization are expressed. Chapter 3 presents the methodology used in the construction of this iteration for the framework. Included in this are the components of the proposed architecture, as well as the specific algorithms used in place for the Cooperative Localization method. Chapter 4 describes the framework’s performance on how it handles the designated task with dynamic operating environment conditions. Also included are the results and analysis generated from the cooperative localization algorithm. This document concludes with a discussion in Chapter 5 on the data collected from the algorithm. Additionally, the considerations made for this framework’s iteration are addressed in the chapter to provide guidance on the further developments needed for full test-bed functionality.

1.2 Research Motivation

Within the last two decades the effects of rapid technological changes have become more evident [1]. This concept of *change* has quickly become a cultural norm and expectation in the years to come. In *America’s Air Force: A Call to the Future* this new norm is said to come with serious implications for the Air Force due to its effect on our operational advantages. These continuously emerging technologies produce new disruptive techniques usable by our adversaries which in hand shorten the lifespan of our advantages. Through *strategic agility*, the *Strategic Vectors for the Future*, and a 30-year plan we hope to address these implications and remove this threat [1]. Of these vectors for the future, one addresses this vulnerability through the continuous pursuit of our own “game changing” technologies. This strategy relies on the development towards advances that amplify the speed, range, flexibility and precision attributes of air power [1]. Two technologies that fall within this spectrum are that

of Unmanned Systems and Autonomy [1]. Development in these areas according to the proposed 30-year plan focus on the extension of our capabilities through the use of such systems in order to protect our airmen. This means less airmen in physical danger while simultaneously increasing our operational capabilities through remote collaboration with our unmanned and autonomous assets. To accomplish this collaboration effectively *Autonomous Horizons: System Autonomy in the Air Force - A Path to the Future* stresses the importance of a *flexible autonomy*, which is one where depending on the situation, will shift anywhere between operating under fully manual and full autonomy [2]. By definition, to operate under full autonomy a system must exhibit the following three principles: *task*, *peer*, and *cognitive* flexibility [3]. Unfortunately, there is yet to exist a platform that fully meets these performance requirements due to each of the varying frameworks and approaches up to present-day falling short. To aid in implementation of the 30-year plan, a robust framework that can evaluate autonomy capabilities is desirable for allowing the Air Force to explore autonomy in UAVs [3].

1.3 Research Objectives

The aim for this research is to contribute towards a sufficiently robust platform to assist in the development of different autonomies. The developed platform is intended to be used for the testing and evaluation of these new autonomies with respect to the three principles of task, peer, and cognitive flexibility. Specifically, the development discussed in this document focuses primarily on evaluating the principle of peer flexibility through the use of cooperative localization. The long term design goal for the framework revolves around the following criteria [3]:

- Well-defined and abstract software elements (components) that require little to no modifications from one evaluation to the next

- Interoperability between these software elements (components)
- Communication capabilities between vehicle-to-vehicle as well as human-to-vehicle
- Exhibition of *task*, *peer*, and *cognitive* flexibility

1.4 Assumptions and Constraints

This document presents the work done that contributes to the long term goal of providing a fully functional testing platform. To reduce the scope and begin the development some assumptions and constraints needed to be made. In the proposed design there exists multiple components with intricate responsibilities. Furthermore, these components operate using large amounts of information passed throughout the framework. Below are some broad assumptions for the framework and the information shared within:

- All measurements and state information transmitted are characterized by normal distributions with a mean value and covariance
- The time between spatial relative observations of vehicles and transmittance to other team vehicles is assumed to be in real-time, and therefore negligible
- Bearing measurements are with respect to the vehicle frame
- Heading measurements are with respect to North

To establish a pattern of inheritance for future iterations some assumptions as to how this information is received impacts some of the design choices behind each component. For rapid development purposes, the framework was designed using MATLAB's Object Oriented environment in which the class definitions for each component drew

from some basic principles from Object Oriented programming. Because of this, some methods used within each Class needed to be modified to handle matlab argument passing. Further assumptions and constraints related to the experiment are discussed in Chapter 3.

II. Background

In this chapter, an overview of fundamental concepts necessary for understanding the functions and implementation of this Autonomy Framework is provided. Additionally, included are the literature relevant to the encompassed topics within this research area. This chapter is organized in the following manner. In Section 2.1 the inner workings and general function for a software framework are explored. Some common approaches for framework design are presented along with their strengths and weaknesses with the intent to provide contrast for this work's taken approach. Section 2.2 examines communication frameworks and their necessity in middleware applications. Some commonly used and standard frameworks are included. Lastly, Section 2.3 introduces the concept of Cooperative Localization; the proposed mission application for testing of peer flexibility on this platform.

2.1 Robotic Frameworks

When it comes to accomplishing tasks, depending on the task at hand, humans can quickly derive solutions. Whether it comes from experience or cognitive ability, the human brain can process the environment around it and come up with an adequate solution. The intricate workings and anatomy of the human brain allows for this parallel processing of environmental information and actuator control. For a robot to accomplish the same task that an intelligent being with a brain is presented with, a collection of tools and modules are necessary. This collection of building blocks are what constitutes a Robotic Framework or Robotic Software Framework (RSF). This definition for a Robotic Framework is not to be confused with Robotic Middleware as there is an important distinction. When it comes to Middleware it is important to think of it as the *glue that holds these modules together*. In essence, it is the

communication infrastructure that connects the different parts of the framework and allows them to function together [4]. In the following subsections some of the more widely accepted Robotic Paradigms, or architecture design patterns, are explored in brief detail. Additionally, some expansion is provided on the control techniques, or arbitration methods, for action generation that some of these design patterns implement.

2.1.1 Deliberative Architectures.

The deliberative approach to framework design represents a top-down philosophy in which there exists a hierarchy of steps for robotic functions. This method is the oldest out of all design patterns and was the focus of Artificial Intelligence Research for over thirty years until about the mid 1980's [5]. Some common aliases for this approach are the *Hierarchical*[6] and the more commonly known *Sense-Plan-Act (SPA)*[5] architectures. In short, this design pattern is autological. The SPA approach represents a methodology in which the system uses sensors to model the environment, creating a *World Model*, algorithms in place generate plans for the robot and then execute relevant actions. This architecture was first demonstrated by the Artificial Intelligence Center (SRI International), through Shakey the Robot [7].

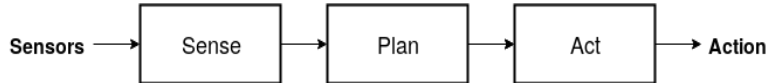


Figure 1. Deliberative Architecture Decomposition [8]

Some issues arise with the SPA approach. For one, the architecture's design depends on an accurate World Model to plan a relevant action/action-set for the system to execute. This causes problems for even relatively simple tasks due to the unrealistic expectation of a static operating environment. Dynamic environments would require the robot to more frequently update the model for the *plan* and *act*

phase to be of more relevance. This can eat up resources and, depending on the level of sophistication in the features of the system, and can also lead to an increase in the number of sensors required for a sufficient World Model. This added complexity increases the latency in the *Sense* phase of the architecture. This was observed in the implementation on *Shakey the Robot*. Of even more consequence is the fact that, the system is unresponsive to its environment during the *Plan* and *Act* phases [9], meaning that any changes in the surroundings would not be taken into consideration during each of these phases respectively. The combination of these shortcomings “results in sub-par performance” when it comes to control output [10].

2.1.2 Reactive Architectures.

Due to the shortcomings of the *Deliberative* approach, a new architecture was needed. The origin of this new paradigm was credited to *Valentino Braitenberg*, who, in the mid-1980’s, presented an architecture that was derived through the inspiration of living organisms. He proposed that machines/robots should not be viewed from an engineering perspective but instead through “psychological language” to “describe their behavior”. The basis of this concept came from Braitenberg’s idea that complex behaviors exhibited by living organisms may be the result of simpler behaviors [11]. His thoughts were that through this different lens, one could shift focus from viewing hardware as a *realization of an idea* to the *idea* itself. As a result, the *Reactive* approach was born [12]. The main benefit from this new approach was that through the elimination of an internal “World Model” being passed into some sort of planning layer, actions could be directly coupled to the information coming through the sensors. This modification to information flow resulted in a reduction to computational latency, thus bringing in the era of reactive architectures. Braitenberg’s *Reactive* approach soon led to the inspiration of a number of derivative architectures,

with each one placing emphasis on certain aspects of the overall concept. Below are a few of these derived architectures.

Subsumption. The *Subsumption* architecture was one of the first *Reactive* approaches prevalent in the 1980's [13]. *Rodney Brooks*, the creator, looked to insects as inspiration for the design. Through observation he deduced that by organizing the reactive layer in a “bottom-up” manner, he could aggregate intelligence upwards and achieve a decrease in computational latency. To do this, he found it necessary for each sub-layer within this *reactive* layer to be organized by levels of competence in which each sub-layer *subsumes* from the layers under it. Additionally, they were to be designed for independent and asynchronous application. By extension, the coupling of sensory information directly to each of these sub-layers dismantles the “World Model” into “sub-states” allowing for the simpler layers to generate simple results [14, 15]. With this design, the higher level and more complex layers (less competent) within the *Reactive* layer are capable of “subsuming” the lower (more competent) sub-layers allowing for more intricate actions to be generated.

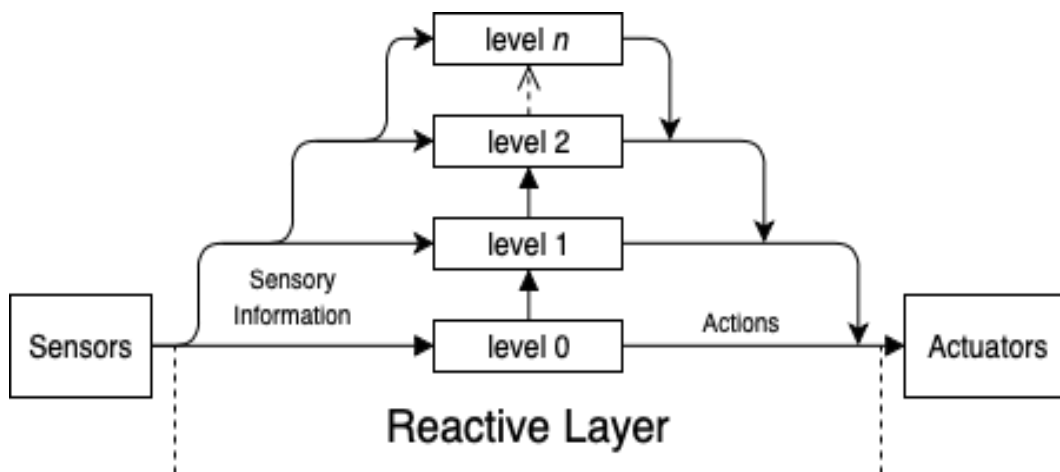


Figure 2. Subsumption Architecture Decomposition [16]

Colony Architecture. Following in the footsteps of *Brooks’* subsumption architecture, Jonathan Connell came up with the *Colony Architecture* in 1989[17]. Just like its predecessor the *Subsumption Architecture* emerged as a cooperative styled approach in which it operated by running on multiple processors/microcontrollers. Although intently reactive, the *Colony Architecture* differed from the hierarchical organizational structure of *Subsumption*. By design, this architecture consisted of a colony of smaller and simpler behaviors that worked collectively in parallel to generate actions. For developmental purposes the parallelization worked well because one could focus on the design, implementation, and troubleshooting of individual behaviors prior to moving onto another. Additionally, if features or new behaviors needed to be added then it was as simple as adding another processor to handle the task. Unfortunately this compartmentalization of behaviors also meant that if the system suffered from hardware failure at a given point, an entire group of behaviors could be affected; rendering the overall system unable to continue functioning. Designing fail-safes or backup plans becomes naturally more difficult in such conditions. [17]

Motor Schema. Originally theorized by *Richard A. Schmidt* in 1975 [18], the Motor Schema theory was proposed as an alternative for the open and closed-loop psychological theories existing at the time regarding motor skills learning in an individual. His alternative was modeled around the notion that “recall memory” should produce movement and “recognition memory” should evaluate the response correctness [18]. In 1993, Ronald Craig Arkin reified this theory as an architecture in robotic navigation [19]. In his version of a reactive approach, each sub-layer within the reactive layer represented a particular simple schema, or *behavior*, such as “wander” or “avoid objects” [11]. Like in Schmidt’s theory, Arkin’s architecture “recalled memory” through environmental stimuli with the use of sensors. This was done without the persistent retention of information about the operating environment, with

the exception of *apriori* knowledge where needed to stay true to a reactive approach. Each of the behaviors was designed so that they could generate an action with the available sensory information. The generated action was in the form of a velocity vector. This design allowed the actions to be combined through vector summation and normalization. The Motor Schema approach was, consequentially, not considered priority-based in regards to arbitration and emerged as one the first “cooperative control” methods [20]. Like the rest of the reactive approaches, this method suffered from cyclical behavior and issues with local minima/maxima due to the lack of retention of world information and deliberation [10, 14].

2.1.3 Early Hybrid Deliberative/Reactive Architectures.

Inspired by previous architectures, some researchers sought new approaches that would improve upon the shortcomings of both the deliberative and reactive paradigms as individual architectures. Using the strengths of both the deliberative and reactive approaches to circumvent their individual weaknesses, hybrid architectures were able to improve upon what each individual approach could not. The general idea behind this approach was that by incorporating deliberative and reactive layers in the overall architecture, one could provide a real-time performance capable of long-term plans.

Circuit Architecture. In her attempt to address problems with traditional reactive approaches, Leslie P. Kaelbling proposed the Circuit Architecture in 1986 [21]. As seen in Figure 3, the name Circuit Architecture comes from a feedback methodology in which generated actions from the action component are provided as feedback to the perception component. One of the main intents behind this architecture was the need for verifying that the generated actions of a reactive architecture were aligned and relevant to the system’s intended effects. The idea is that without this verification process on the back-end a system is not truly reactive, hence the need

for some feedback [21].

Using *Brookes'* approach as influence, the architecture was designed with perceptual and behavioral robustness. The architecture took advantage of the possibility for redundant sensory information from multiple sensor types. By integrating the sensory information from these multiple sources into a structure representing something similar to a World Model, the perception component aimed to provide sufficient information to trigger actions in accordance with the quality of available information. Through the combination of the *tick* and the feedback coming from the action component, the architecture could also provide behavioral robustness. As seen in Figure 3, the system could validate whether its actions were having their intended effects on the environment by feeding back information related to the generated actions. This information would include but not be limited to: a signal referencing the current inability to formulate plans, necessary data requiring collection for the completion of higher level plans, and current operational focus due to survival needs. Kaelbling acknowledged the importance of sensor failure detection and its effect on the integrity of the generated World Model but did not propose a solution in the Circuit Architecture's original design [21].

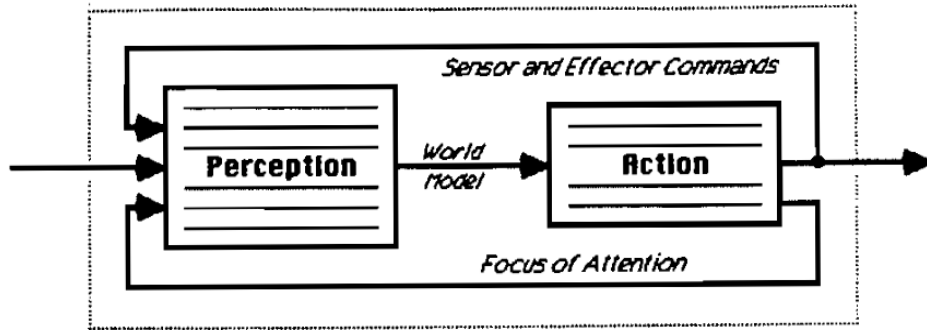


Figure 3. Circuit Architecture Perception-Action Decomposition [21]

2.1.4 Centralized and Distributed Architectures.

In the early stages of robotics and autonomy the approaches taken for architecture design relied heavily on the technology available to the developers at that time. As mentioned above, the *deliberative* and *reactive* approaches that emerged focused primarily on innovation between perception and action with the general end-goal being to generate the most effective set of actions and execute them [22]. This led to the layered approaches referred to as the “top-down” and “hierarchical” architectures presented in the previous sections. Unfortunately even with the early stage hybrid architectures that emerged to circumvent the individual shortcomings of both deliberative and reactive approaches, they were still generally inhibited by their organizational structure. In 1997, Jonathan Rosenblatt [22] proposed that instead of imposing a hierarchical structure to achieve a hybrid architecture, one could use multiple modules to concurrently share control of the robot and generate actions through arbitration [22]. This general idea applied to both Centralized and Distributed Architectures with the main difference being in the distribution of control and planning throughout the framework.

D.A.M.N.. *Rosenblatt’s* first demonstration of a distributed architecture was in his *Distributed Architecture for Mobile Navigation*, or DAMN [22]. Like some previous reactive systems, DAMN was designed as a behavior-based architecture with its distinction laying in its method for action generation. Unlike the Colony Architecture, DAMN utilizes an Arbiter module that receives votes from its behaviors and then uses a Command Fusion arbitration technique, which consists of fusing behavior outputs to generate a set of commands for the controller that best satisfies the overall goal of the system [17, 22]. This methodology allows for the consideration of multiple constraints stemming from each behavior asynchronously.

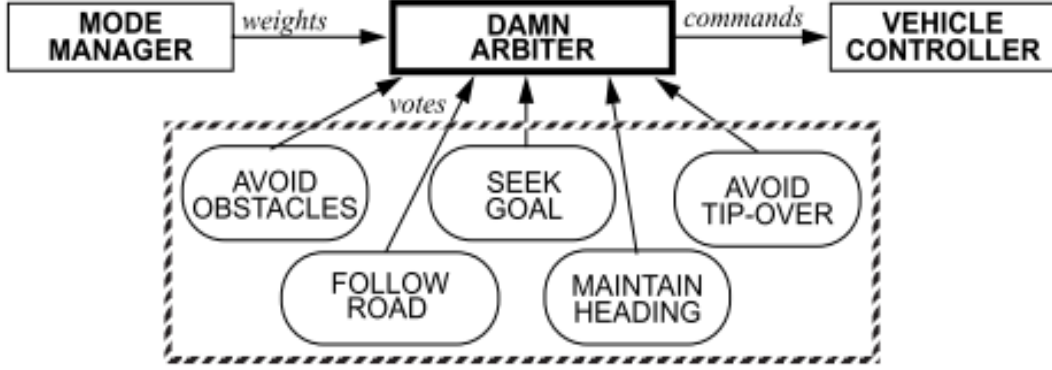


Figure 4. Overall structure of DAMN [22]

2.1.5 Arbitration Techniques.

Prior to the development of *Centralized* and *Distributed* architectures, each approach typically was designed with a particular arbitration method in mind. Although not all approaches actually included an Arbiter module, actions were still generated through some form of arbitration. When it came to performance, each architecture generally did well around the design criteria, but the effectiveness did not always hold true outside of their original scope. The practice of designing architectures around their design criteria implicates the level of robustness for varying operating conditions due to lack of compatibility with different arbitration methods. Below are some of the more popular arbitration methods used, some of which are present in the previous architectures. Each of these methods follow the standard practice of enabling arbitration through the use of a weighted system for generated actions. The arbitration techniques then take these actions and their associated weights as inputs to provide an output consisting of desired states for the controller.

Priority-Based. Also known as the Winner-Take-All (WTA) arbiter, the Priority-Based arbiter simply returns the action with the highest vote as a command to the robot controller.

Vector Summation. Often used in systems with velocity control, the Vector Summation arbiter takes in the set of outputs from each behavior and their associated weights. This input into the arbiter then combines the inputs to generate some form of a weighted sum of actions, which is then fed to relevant/compatible actuators.

Subsumption. This arbiter is used when there exists a set of competency rules for arbitration within the architecture. The vertical decomposition of behaviors used in the Subsumption Architecture allows for a system of checks-and-balances in which certain behaviors can inhibit others based on their competency level. One such metric for competence can be time of arrival for action recommendation from behaviors, as seen in an experiment by Taylor Bodin [9]. This set of rules can allow the arbiter to select or fuse actions based on a first-come-first-serve basis [9].

Command Fusion. As seen in the *DAMN* architecture, the Command Fusion arbiter fuses some of its input data. The distinction lies in that the action generated is not that of a vector sum but instead through a priority-based selection; what is actually summed is the collection of weights from each behavior. Each behavior weighs all possible relevant commands and then outputs this vector to the arbiter. The arbiter then takes all of the outputs from each behavior and then sums them in correspondence to the relevant actuator. The action associated to highest weight is then passed on to that actuator [22].

Utility Fusion. Utility Fusion was introduced as an alternative to priority-based arbitration methods. Instead of proposing actions with associated votes, behaviors indicate the utility of potential states to the arbiter [23]. To determine the necessary commands for the actuator(s), the arbiter then combines the utilities fed into it and attempts to maximize this information to provide an optimal state. This

arbitration scheme allows for action generation without the need of a World Model, giving the system running it the capability for more “intelligent decision-making” [23].

2.2 Middleware

As stated in Section 2.1, *Robotic Middleware* are what is considered to be the glue that holds robotic modules together [4]. The interaction between these different software and hardware modules is made possible through the variety of services provided in each middleware suite. In distributed architectures, these services fulfill a critical role in not only information/data transport but also in portability, reliability, and complexity management [24]. The usefulness of middleware extends beyond robotic framework design. In a broader scope Middleware can be categorized into one of three groups: application-specific, information-exchange, and management/support [24]. When it comes to picking a middleware suite for a robotic framework there are a number of options available, unfortunately not all are under active development. In this section a few of the more widely used and actively developed middleware are discussed.

2.2.1 Robotic Operating System.

Among the middleware in active development the Robotic Operating System (ROS) is one of the more widely used for robotics systems due to its extensive documentation and community support available. Originally developed in 2007 at the Willow Garage, ROS was created in support of the Stanford Artificial Intelligence Robot (STAIR). The goal behind the subsequent development for this middleware was to encourage the reuse of code and collaboration among its users. Although this middleware itself is not a real-time framework, its strength lies in the large user com-

munity, making it a powerful tool for research and rapid development. The framework itself is open-source and functions as a meta-operating system for a robot, providing all of the tools and services needed for operating a wide variety of robots. The filesystem design of ROS consists of a distributed framework of processes, or *nodes* [25]. These processes are what the community calls packages and stacks. Some of the tools provided allow for the “obtaining, building, writing, and running of code across multiple computers” [25].

2.2.2 Google Protocol-Buffers.

Developed by Google internally and then publicly released in 2008 Google Protocol-Buffers, *Google Protocol-Buffers* was created for the purpose of serializing structured data. This middleware is language-neutral and platform-neutral allowing it to be implemented in a variety of different programming languages on an unrestricted set of platforms. The message structures consist of a simple format in which the user creates a **.proto** file that includes the message contents in a user-defined set of structures. The data in each of these structures consists of a fieldname and a value. This value can be of either boolean, integer/floating-point number, strings, raw bytes, or even references to other buffer message types [26]. Optional properties can be defined for each of these fields allowing the user to specify them as optional, required, and repeated [26]. This particular middleware is a worthy choice when the user cares for a simpler structuring of data, smaller/more lightweight messages, and low-latency messaging.

2.2.3 Lightweight Communication and Marshalling.

Initially developed in the Computer Science and Artificial Intelligence Laboratory of the Massachusetts Institute of Technology (MIT) in 2009, the Lightweight Com-

munications and Marshaling (LCM) libraries were created for message passing and data marshalling [27]. These libraries were originally made with the intent to simplify the development of low-latency communication between modules. An attractive area for such application was in the field of robotics where real-time performance can be highly desired. The features found in the set of LCM libraries are a message passing system, tools for the logging/playback of message data, tools for “real-time” analysis, and platform/language-independent specification [27]. The messaging system for data passing consists of a publish/subscribe model in which messages containing user desired information can be “published” by any system module into a user’s network and then “subscribed” to by any module on the same network to receive the published information. The structuring of these messages is similar to the method used in Google Protocol-Buffers. These message structures are then saved in **.lcm** files to be compiled for use. The toolkit and design allows for the inspection and modification of messages on the network in the event that performance increase is needed.

Additional Middleware. Some other honorable mentions for middleware in robotics include but are not limited to the following: CLARAty [28, 29, 30, 31, 32], OPRoS [33, 34, 35, 36], Orocos [37], Player [38, 39, 40], and YARP [41, 42]; some of which are no longer in active development [24, 4]. Table 1 [4] compares a few of these middleware in order to provide some insight on their compatibility with different operating systems and languages.

Table 1. Middleware Comparison Table from *Robotic Frameworks, Architectures, and Middleware Comparison*[4]

Robotic Frameworks	Operating System	Programming Language	Open Source	Distributed Architecture	HW Interfaces and Drivers	Robotic Algorithms	Simulation	Control/Real-Time Oriented
ROS	Unix	C++, Python, Lisp	✓	✓	✓	✓	~	X
HOP	Unix, Windows	Scheme, Javascript	✓	✓	~	X	X	X
Player/Stage/Gazebo	Linux, Solaris, BSD	C++, Tcl, Java, Python	✓	~	✓	✓	✓	X
MSRS (MRDS)	Windows	C#	X	✓	~	X	✓	X
ARIA	Linux, Win	C++, Python, Java	✓	X	✓	✓	X	X
Aseba	Linux	Aseba	✓	✓	✓	X	~	✓
Carmen	Linux	C++	✓	✓	✓	✓	✓	X
CLARAty	Unix	C++	✓	✓	✓	✓	X	X
CoolBOT	Linux, Win	C++	✓	✓	~	X	X	X
ESRP	Linux, Win	?	X	✓	✓	✓	X	X
iRobot Aware	?	?	X	?	✓	?	X	?
Marie	Linux	C++	✓	✓	✓	X	X	X
MCA2	Linux, Win32, OS/X	C, C++	✓	✓	✓	X	X	✓
Miro	Linux	C++	✓	✓	✓	X	X	X
MissionLab	Linux, Fedora	C++	✓	✓	✓	✓	✓	X
MOOS	Windows, Linux, OS/X	C++	✓	~	✓	✓	X	X
OpenRAVE	Linux, Win	C++, Python	✓	X	X	✓	✓	X
OpenRDK	Linux, OS/X	C++	✓	✓	✓	X	X	X
OPRoS	Linux, Win	C++	✓	✓	✓	✓	✓	X
Orca	Linux, Win, QNX Neutrino	C++	✓	✓	✓	~	X	X
Orocos	Linux, OS/X	C++	✓	✓	✓	✓	X	✓
RoboFrame	Linux, BSD, Win	C++	?	✓	✓	X	X	X
RT Middleware	Linux, Win, CORBA platform	C++, Java, Python, Erlang	✓	✓	✓	X	X	X
Pyro	Linux, Win, OS/X	Python	✓	X	✓	✓	✓	X
ROCI	Win	C#	✓	✓	X	X	X	X
RSCA	?	?	X	X	✓	X	X	✓
ROCK	Linux	C++	✓	?	✓	✓	X	✓
SmartSoft	Linux	C++	✓	✓	X	X	X	X
TeamBots	Linux, Win	Java	✓	X	✓	✓	✓	X
Urbi (language)	Linux, OS/X, Win	C++ like	✓	X	✓	X	X	X
Webots	Win, Linux, OS/X	C, C++, Java, Python, Matlab, Urbi	X	X	✓	X	✓	X
YARP	Win, Linux, OS/X	C++	✓	✓	✓	X	✓	X

2.3 Cooperative Localization

Cooperative Localization refers to the process in which multiple systems work together to help obtain/improve position information about either themselves or a foreign entity. The purpose behind accomplishing such a task through cooperative methods revolves around the idea that more perspective results in better estimates for solutions. Generally when it comes to real-life applications, a higher level of accuracy greatly benefits operational performance. Unlike human operated systems, robotic platforms exhibiting autonomy stand to benefit the most from such accuracy due to the lack of secondary controls being present (human control). In this section some of the fundamental concepts are provided for two of the more commonly used tools in cooperative localization: filtering methods and factor graphs [43, 44, 45, 46, 47, 48].

2.3.1 Bayesian Filters.

Among the fundamental filtering methods used in state estimation, the Bayes Filter stands as one of the most important due to its presence and influence on other filters. The Bayes filter is a probabilistic approach for estimation that works through recursion. The filters that are derived from this recursive bayesian estimation approach all share a common methodology that is present in their predecessor. In short, the filter works through a sequence of steps in which a prediction is made about a particular upcoming measurement, using some model about the system. Measurements are taken and compared to the estimates. At this point, adjustments/updates to the state are then made. These new adjustments are based off of statistical analysis of the model itself and of the measurements obtained.

The entire process involved in the Bayes Filter is considered to be *Markhov*, meaning that at any given point in time the current event depends only on the state obtained in the previous event. This relation implies that the current model can be

considered to contain historical information about all the previous models. Additional information regarding Bayesian filtering can be found in [49, 50, 51, 52, 44], however this thesis primarily utilizes factor graphs for statistical inference.

2.3.2 Factor Graphs.

In 2001, Frank R. Kschischang introduced *factors graphs* as an approach for dealing with complex functions consisting of many variables [53]. These factor graphs were *bipartite*, meaning that they represented global functions as a product of smaller, more local, functions. These local functions were typically multivariate and a subset of the total variables present in the global function.

Let a global function be represented by $g(x_1, \dots, x_n)$ where the set $\{x_1, \dots, x_n\}$ refers to the variables present in the model. Due to the bipartite nature of these factor graphs, this global function can be represented by

$$g(x_1, \dots, x_n) = \prod_{j \in J} f_j(X_j) \quad (1)$$

where $f_j(X_j)$ refers to a single factor, or function, consisting of a subset of variables from the set $\{x_1, \dots, x_n\}$. To illustrate further, we can represent this graphically as seen in Figure 5, where *variable nodes* are represented by x_n and *factor nodes* are represented by f_n .

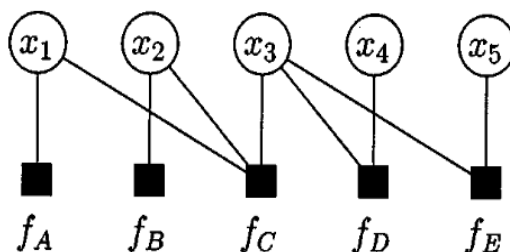


Figure 5. Factor graph for the product of $f_A(x_1)f_B(x_2)f_C(x_1, x_2, x_3)f_D(x_3, x_4)f_E(x_3, x_5)$ from *Factor Graphs and the Sum-Product Algorithm* [53]

In short, these factor graphs outline the fundamental relationships between variables and a set of factors. Inferences can then be made about each factor, allowing the use of algorithms that work well on trees such as the Bayesian Tree.

The usefulness of factor graphs extends into the world of robotics where perception plays a critical role, as seen in Section 2.1. Making of inferences about the operating environment plays a critical role in evaluating the effectiveness of actions. By design, factor graphs prove to be useful because of how they are structured and their compatibility with inference-making algorithms. This utility has led to popularity in mapping algorithms where position information can be represented through variable nodes and perception through factor nodes in a factor graph. By inferring about the global function through the use of measurements and tree structures, corrections to drifting error can be made to not only the current state but all previous states in the graph as well.

As stated previously, a factor graph corresponds to the factorization of a global function. Using probability theory, we can express the global function as a *probability density function (PDF)*. The global function can be factored into the smaller components, using (1), with each factor $f_j(X_j)$, corresponding to a particular measurement. By turning this into a maximization problem,

$$\operatorname{argmax}_{X_j} \prod_j f_j(X_j) \quad (2)$$

the optimal solution for the measurement $f_j(X_j)$ can be found by looking for the right combination of X_j , where X_j represents all of the known vehicle poses and landmarks in that factor, that maximizes the probability of that particular measurement. Simi-

larly, if Gaussian noise is assumed, then (2) can be reduced to,

$$\operatorname{argmin}_{X_j} \sum_j |h_j(X_j) - Z_j|_Z^2 \quad (3)$$

where $h_j(X_j)$ represents the predicted measurement according to a generative sensor model, and Z_j represents the measurement from the sensor itself. Additionally, if the assumption is made that the generative model $h_j(X_j)$ is linear, and this is supplemented through frequent linearization, then (3) can be represented as,

$$\operatorname{argmin}_{X_j} |AX_j - b|_Z^2 \quad (4)$$

where A is a large matrix with each row containing the variables in X_j that corresponding to a particular factor $f_j(X_j)$. Due to the design of the original factor graph, this matrix A exhibits the properties of a sparse matrix, allowing for the use of more efficient optimization algorithms. Incremental Smoothing and Mapping (iSAM2) is one of these algorithms which will be used in the Cooperative Localization piece of this framework. [52, 54, 55]

III. Methodology

This chapter reviews the initial architecture design for the proposed Autonomy Framework. The organizational structure is depicted in Figure 6 and is heavily based on the Hybrid Architecture for Multiple Robots (HAMR) [56]. For development purposes the framework was built using MATLAB’s R2018a object-oriented programming environment. The use of MATLAB for the construction of this framework allows for the application of standard object-oriented design patterns found in other languages such as C++ and JavaTM. To validate the principle of *peer flexibility* in autonomy,

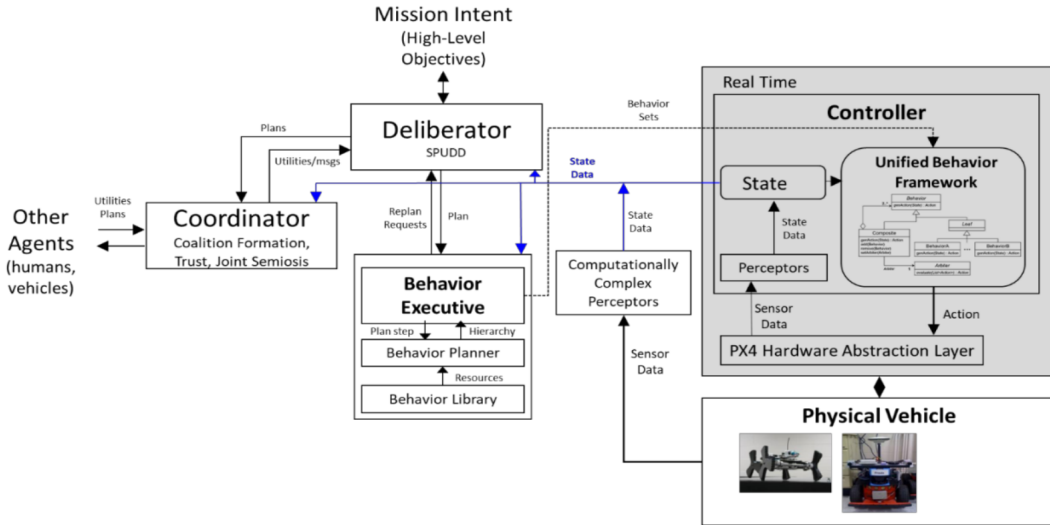


Figure 6. Hybrid Architecture for Multiple Robots [3]

the focus in the framework’s development was for eventual implementation on Multi-Robot Systems. All modules within the framework communicate with each other via LCM channels using a publish/subscribe approach [57]. This design allows for the subscription of a single topic by multiple modules in need of the same information.

In Section 3.1 a breakdown for the internal workings of this HAMR-based framework are laid out. Some expansion on the algorithms used and class definitions are provided. Section 3.2 discusses the objective for this framework used to help validate

peer-flexibility. The software packages used and algorithms developed for execution of this mission are also presented. Lastly, Section 3.2 overviews the experimental runs used for determining the effectiveness of the framework’s design for Multi-Robot Systems.

3.1 HAMR

As seen in Figure 6, HAMR builds upon a traditional three layer architecture through the addition of a *Coordinator* layer. By abstracting a layer for the interaction of other agents in a Multi-Robot System, the *Deliberator* can be left to perform its high-level tasks [56] without the added complication of handling a multi-agent environment. This also reduces the communication overhead [58] handled by the Deliberator due to the streamlining and compartmentalizing of information which gets communicated in and out of a coordination component in a Multi-Robot System. In the subsections below the roles assigned to each layer are expanded in addition to some of the important properties and methods used for their class definitions.

3.1.1 Deliberator.

Like previously stated, the *Deliberator* layer in this framework is responsible for performing high-level reasoning tasks. These tasks consist of anything between task decomposition, task allocation between different agents, and mission planning [56]. Due to the multi-agent nature behind the purpose of this work, some additional requirements are needed for proper deliberation [59, 60], therefore to keep the focus strictly on multi-vehicle coordination, some of the complexity was taken from this iteration of the Deliberator. Although the intricacies of the Deliberator were intentionally reduced, the same structure for information flow was kept with respect to Figure 6 to stay true to the Deliberator’s responsibilities. All of these responsibili-

ties are accomplished through the provision of state information from the *Controller* layer, utilities and messages about the agents relevant to the operating environment from the *Coordinator* layer, and mission updates or re-plan requests from the *Sequencer* layer. Using this information the robot is able to take the *Mission Intent*, or High-Level objectives, and send plans to the Coordinator for agent cooperation and the Sequencer for behavior planning.

3.1.2 Controller.

Within the *Controller* layer of this Autonomy Framework exists a number of smaller components working together to accomplish physical vehicle actuation. Listed below are breakdowns of the different components that make up this layer as shown in Figure 6.

3.1.2.1 Unified Behavior Framework.

In parallel with a modified three-layer architecture, this autonomy framework operates under a behavior-based architecture. Typically, these types of architectures are lacking in robustness and are better suited for tightly tuned environments [61]. Deviating from such environments requires augmentation to accommodate more functionality. Unfortunately, adding this functionality also impacts control complexity and results in degraded performance. To circumvent this degraded performance, the Unified Behavior Framework (UBF) [62] is implemented as a behavior architecture for the autonomy framework’s controller. Through the use of a composite and strategy pattern the UBF is able to break down tasks assigned to the controller, make assignments and enable autonomous behaviors [63]. These assigned tasks come from the *Sequencer* layer in the form of a *Behavior Set*, relevant to the current mission plan. This set is dynamic and can change depending on the relevant behaviors determined

from the Sequencer. Consequently the changes in the behavior set are low frequency and only changes when there are updates to the current mission. This design choice allows the controller to operate under the highest frequency possible with the intent of real-time operation. Within the UBF, this Behavior Set is then instantiated, allowing for desired states and associated priority values to be passed into the Arbiter for action selection via the *Winner-Take-All* method.

There are a couple important class definitions that should be discussed here. Included below are a few of the class definitions paramount to the framework along with an associated functional description.

Arbiter. The *Arbiter* Base Class, or abstract class in MATLAB, represents a class definition to be used when creating different Arbiter derived classes. In this particular framework, the *Winner-Take-All* Arbiter inherits the properties and methods from this base class and, applies arbitration via priority selection. The derived Arbiter classes take in an action list with associated priority values and generate a desired output for the UBF in the form of a desired system state.

UBF. The *UBF* Class is a Base Class meant for taking the current Behavior List provided by the Sequencer and sending a desired state to the Hardware Abstraction Layer, which is then translated to wheel/motor commands to actuate the vehicle. The class takes the input Behavior List and instantiates a structure of Behavior Objects. This structure pulls in the state information required by each individual behavior to generate a set of actions along with an associated priority vote. This information is then passed into the Arbiter of choice within the UBF, ultimately providing the system with an desired state and action. This Behavior List changes at a low frequency and is only modified when the Sequencer pushes a different list to the Controller.

3.1.2.2 State.

The *State* component of the Controller layer is responsible for handling all state data that comes from onboard sensors and is necessary for the behaviors to generate an Action. Instead of building a World Model, this component compartmentalizes the state data into smaller components ranging from position, velocity, attitude, and peripheral information to allow for independent updates at independent frequencies. The intent behind this is to be able to provide only the necessary state information requested from the different components within the framework at a higher frequency.

3.1.2.3 Perceptors.

The *Perceptor* component represents the collection of different onboard sensors and estimation algorithms. Within this framework the perceptors are divided into two categories: simple perceptors and computationally complex perceptors. The distinction between the two comes from the individual perceptors impact on the “real-time” performance of the controller. Devices that provide simple information such as position, velocity, and attitude are capable of updating the State component without significant impact towards the latency of Action generation. The category of Computationally Complex Perceptors refers to algorithms or sensor modules that provide state information at a low frequency, typically through the processing of data-intensive algorithms, like image processing, for example.

3.1.2.4 Hardware Abstraction Layer.

The *Hardware Abstraction Layer* within the Controller represents the component in the robot that interprets the Actions generated by the UBF and implements them on the onboard actuators. The Hardware Abstraction Layer will also update the State component with any sensor or derived information, such as position, attitude,

and velocity. For the purpose of this framework, the Hardware Abstraction Layer is embodied in the Pixhawk2 autopilot. That can be either the software (used in virtual world) or hardware (real world) version.

3.1.3 Sequencer.

The *Sequencer* layer of this framework is where the facilitation of task management between the Deliberator and Controller layers occurs. This layer plays a vital role in the breaking down of the tasks received from the Deliberator to provide the Controller with the tools necessary for efficient operation. In addition, the Sequencer also provides feedback to the Deliberator regarding the status of the agent’s current assigned task in the event that there is a need for re-planning. This abstraction allows for the Deliberator and the Controller to operate continuously and asynchronously at their own frequency with interruptions/updates occurring only when needed. Additionally, it also removes the need for the Deliberator to be fully aware of the agent’s entire state, limiting this knowledge exclusively to information critical for High-Level Operational awareness. These responsibilities are accomplished through the Sequencer’s three internal components: the *Behavior Executive*, *Behavior Planner*, and the *Behavior Library*.

3.1.3.1 Behavior Executive.

The *Behavior Executive* exists on the front end of the Sequencer. This component is responsible for handling the interaction between deliberation and operational control. In this framework, the Behavior Executive takes the generated tasks from the Deliberator in the form of a messages. These messages allow for the other components in the Sequencer to further break down the assigned tasks into a workable toolkit for the Controller. Once the task received is broken down and interpreted, a

message structure containing a determined list of Behaviors and their assigned weight values/functions is passed into the Controller layer for use. The Behavior Executive is also the component responsible for receiving current state information relating to the agents operational status in regards to situations such as hardware failure, task failure, task completion, or any significant change in data [56]. This information is then either used to produce a new behavior list for the Controller or to request re-planning from the Deliberator.

3.1.3.2 Behavior Planner.

The *Behavior Planner* is the component responsible for breaking down the currently assigned tasks and generating a workable toolkit for the Controller. This toolkit consists of a set of behaviors determined to be necessary for operations pertaining to the assigned task. Additionally, this is where the prioritization of behaviors occurs. The priority values/functions determined for these behaviors are based on the type of task at hand. The purpose of this step is to provide the Controller with sufficient information to allow for proper arbitration in action generation.

3.1.3.3 Behavior Library.

The Behavior Library consists of all the behaviors available to the system. This component is where the Behavior Planner looks to for determining what tools are available for generating a tool-kit for the Controller. In this framework, the library construct itself is represented by a directory and not an actual class definition. This directory includes all of the behaviors available in the framework as individual class definitions. By doing this, the library can be dynamic without having to redefine the entirety of it whenever a new behavior is created for the framework. For this work the Behavior Library consists of only four behaviors necessary for the particular arbi-

tration method used: *AvoidOthers*, *FollowObject*, *FollowPath*, and *GoToXY*.

AvoidOthers. The primary focus of the *AvoidOthers* behavior is to avoid collision with team vehicles, unidentified vehicles, and/or other obstacles present in the environment. Its priority value is defined by a function that relates the current objects detected within the operating environment and the agent's current position and path.

FollowObject. The *FollowObject* behavior's primary focus is on the detection and pursuit of unidentified vehicles within the agent's Field of View (FOV). This behavior's priority value is defined as a static number which is dependent on the currently assigned mission.

FollowPath. The *FollowPath* behavior is responsible for generating and executing a patrol path within an area of interest or operating environment. This behavior's priority value is static and is not dependent on the currently assigned mission.

GoToXY. The purpose of the *GoToXY* behavior is to provide an X-Y coordinate for a location of interest to the agent's actuators. Like the *FollowPath* behavior, its priority value is static and is not dependent on the currently assigned mission.

3.1.4 Coordinator.

Having a layer in Multi Robot Systems that is responsible for the coordination of all agents available provides this network the ability to retain each of the individual agent's capabilities in addition to asynchronous and complementary operation [58]. In this framework the *Coordinator* layer is what aims to provide such cooperative

capabilities. Within this layer exist the collaborative features necessary in a network of robot and/or human systems such as: the formation of teams or coalitions, the sharing of information of interest, the establishment of trust within the ranks, and joint tasks such as *Cooperative Localization*.

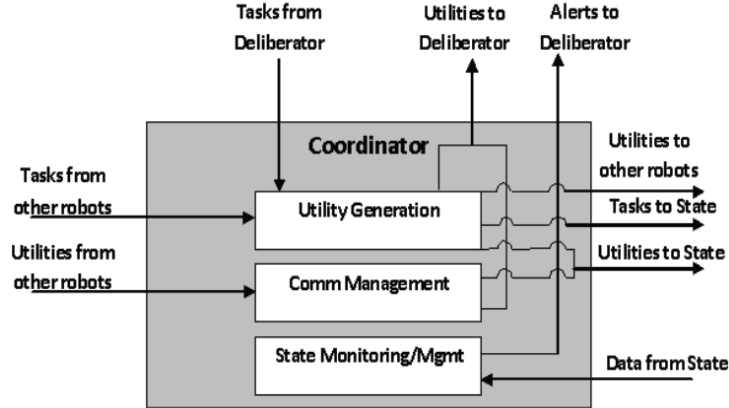


Figure 7. Internal Structure of the Coordinator [58]

In a distributed network of robots where there is no team leader, the Coordinator layer plays a critical role in assisting joint deliberation due to the differences in each agent’s capabilities. These differences can range from the physical properties of the agents in potentially heterogeneous networks to simple operational capabilities due to being in the right place at the right time. In a centralized network of agents where there exists a team leader, the Coordinator acts much less as a team deliberation component like in its distributed counterpart, and instead retains focus on communication and information sharing among the collective. For this work, a centralized approach is used to reduce complexity for the network given the particular mission of interest.

The role of the Coordinator in the Team Leader is to act as the central hub for task designation among the agents and information collection for the Cooperative Localization algorithm. In contrast, the Coordinator layers that exists on the rest of the team members are responsible for receiving tasks from the leader in relation to the current mission and sharing state information of interest among the network for Behavior purposes.

To accomplish coalition formation, a broadcast approach is implemented in which team members entering the operating environment wishing to collaborate with the Leader must publish a message to a secure channel. This channel is under constant surveillance from the Leader in order to allow for the addition or removal of agents from the Team Roster.

3.2 Objective

As stated in Chapter 1.3, to accomplish the goal of validating the principle of peer flexibility a simple scenario was created. The objective of this scenario is to demonstrate the ability of the framework to adjust the internal task assignments

based on the resources available within the operating environment. A simple patrol scenario was designed where conditions of the environment would have a direct effect on the behaviors executed by the agents present and on the ad-hoc network. In this mission an agent(s) is tasked with patrolling a region of interest with the purpose of discerning whether any foreign entities present are friendlies or potential targets of interest. If targets of interest are detected and the resources are available, localization of that target becomes of the utmost priority. To accomplish this, the mission is split into two simple phases: *Patrol* and *Localize*.

1. *Patrol*: The *Patrol* phase of this mission consists of the agent(s) patrolling a designated region of interest through a self-generated path. If multiple agents on the team are present then the region is distributed among all team members with each agent being responsible for patrolling exclusively their own designated zones. Entry of new team members into the environment prompts a redistribution of zones within the region. The *Patrol* phase ends and the team proceeds to the *Pursuit* phase when **ALL** of the following conditions are met:

- A Target of interest enters the operating environment
- There exists sufficient team members (three) actively patrolling in the operating environment

2. *Pursuit*: The *Pursuit* phase of this mission consists of the active pursuit and localization of a target present within the operating environment. This pursuit is carried out by the three team members to provide necessary information to the team leader for the Cooperative Localization algorithm of choice. The *Pursuit* phase ends and team reverts to the *Patrol* phase under **ANY** of the following conditions:

- The Target of interest exits the operating environment

- A loss of a vehicle occurs causing the number of capable vehicles for pursuit to drop below 3

3.2.1 Localization Algorithm.

A Pose Graph Cooperative Localization approach is used to accomplish the localization piece of the above scenario. Like previously mentioned in Chapter 2.3.2, Factor Graph approaches are methods which make inferences about the environment through the use of variables and functions of factors. Making inferences about each factor in relation to the environment allows for the use of sophisticated algorithms that work well on Bayesian Trees. A nonlinear optimization method known as iSAM2 is used to implement the localization algorithm in this work [55]. Through the use of incremental variable re-ordering and frequent re-linearizations, iSAM2 is able to circumvent restructuring of the entire tree built from the generated Factor Graph. This approach saves on computation time allowing for a quicker yet high quality Factor Graph approach in minimizing measurement error.

The Georgia Tech Smoothing and Mapping (GTSAM) library version 3.2.1, which is the parent library for iSAM2, is used in conjunction with MATLAB. GTSAM provides the class definitions necessary for constructing Factor Graphs in MATLAB and applying optimization algorithms on these objects. Algorithm 1¹ depicts the a general algorithm for applying Cooperative Localization. In the algorithm, G represents a factor graph object which inherits its class definition from the GTSAM *NonLinearFactorGraph* class. The variables i and $targ$ are to GTSAM *key* objects which identify the team members and target respectively. Each team vehicle pose and bearing measurement is defined by ${}^i\mu_{xy}$ and ${}^i\mu_\rho$ while ${}^i\Sigma_{xy}$ and ${}^i\Sigma_\rho$ corresponds to each

¹Vehicle poses are 2D, meaning ${}^i\mu_{xy}(t) = (x_{pos}, y_{pos}, \psi)$ and ${}^i\Sigma_{xy}(t) = diag(\sigma_x, \sigma_y, \sigma_\psi)$ for vehicle i at time t

pose and bearing measurement's covariance matrices:

$${}^i\Sigma_{xy} = \begin{bmatrix} {}^i\sigma_x^2 & 0 & 0 \\ 0 & {}^i\sigma_y^2 & 0 \\ 0 & 0 & {}^i\sigma_\psi^2 \end{bmatrix} \quad {}^i\Sigma_\rho = {}^i\sigma_\rho^2 \quad (5)$$

Algorithm 1 General Algorithm for Pose Graph Cooperative Localization

```

1: input:  $G, O, i, targ, t, {}^i\mu_{xy}(t), {}^i\Sigma_{xy}(t), {}^i\mu_\rho(t), {}^i\Sigma_\rho(t)$ 
2: output:  $G$ 
3: // initialize factor graph and optimizer
4: import gtsam.*
5:                                     ▷ import gtsam libraries for use
6:  $G \leftarrow NonlinearFactorGraph$ 
7:  $O \leftarrow gtsam.Optimizer(optimizerParams)$ 
8:  $poseEstimates = Values$ 
9:                                     ▷ define poses as a gtsam Values object
10: while  $localization = true$  do Localization Algorithm
11: // Add factors
12:  $poseEstimates.insert(i, targ, {}^i\mu_\rho(t), {}^i\Sigma_\rho(t))$ 
13:                                     ▷ add estimates
14:  $G \leftarrow add(BearingFactor2D(i, targ, {}^i\mu_\rho(t), {}^i\Sigma_\rho(t)))$ 
15:                                     ▷ add bearing measurements
16:  $G \leftarrow add(BetweenFactorPose2(i_{t-1}, i_t, {}^i\mu_\rho(t), {}^i\Sigma_\rho(t)))$ 
17:                                     ▷ add odometry measurements
18: // pose graph optimization
19:  $O \leftarrow update(newFactors, newEstimates)$ 
20: return:  $G, O$ 
21: end while

```

3.2.2 Assumptions.

For this experiment, some assumptions are made regarding the sensors and states of the vehicles to simplify the modeling. Table 2 lists all state variables collected and their relative information

- All target vehicles are autonomous and move according to a velocity motion model [64]
- All team vehicles have knowledge of their position at all times. This information is represented by an East, North, and Altitude position. Each variable is characterized by a normal distribution.
- All team vehicles receive attitude information: Roll (ϕ), Pitch (θ), and Yaw (ψ). Each variable is characterized by a normal distribution.
- All team vehicles receive velocity information: $v_x, v_y, v_z, \omega_\phi, \omega_\theta, \omega_\psi$. Each variable is characterized by a normal distribution.
- All team vehicles receive acceleration information: $a_x, a_y, a_z, \alpha_\phi, \alpha_\theta, \alpha_\psi$. Each variable is characterized by a normal distribution.
- All team vehicles are outfitted with a fixed camera, providing them with a bearing measurement $\rho \sim N(\mu_{\rho n}, \sigma_{\rho n}^2)$ when the target exists in the cameras FOV
- Communication between vehicle nodes (UGV_1, UGV_2) and central node (UAV_1) is assumed to be in *real-time*

Table 2. State Variables

Parent Structure	Variables/Factors	Characterization	Values	Units
Position	x_n	$\sim N$	$\mu_{x_n}, \sigma_{x_n}^2$	x_n
	y_n		$\mu_{y_n}, \sigma_{y_n}^2$	
	z_n		$\mu_{z_n}, \sigma_{z_n}^2$	
Attitude	ϕ_n	$\sim N$	$\mu_{\phi_n}, \sigma_{\phi_n}^2$	ϕ_n
	θ_n		$\mu_{\theta_n}, \sigma_{\theta_n}^2$	
	ψ_n		$\mu_{\psi_n}, \sigma_{\psi_n}^2$	
Velocity	\dot{x}_n	$\sim N$	$\mu_{\dot{x}_n}, \sigma_{\dot{x}_n}^2$	\dot{x}_n
	\dot{y}_n		$\mu_{\dot{y}_n}, \sigma_{\dot{y}_n}^2$	
	\dot{z}_n		$\mu_{\dot{z}_n}, \sigma_{\dot{z}_n}^2$	
	$\dot{\phi}_n$		$\mu_{\dot{\phi}_n}, \sigma_{\dot{\phi}_n}^2$	$\dot{\phi}_n$
	$\dot{\theta}_n$		$\mu_{\dot{\theta}_n}, \sigma_{\dot{\theta}_n}^2$	
	$\dot{\psi}_n$		$\mu_{\dot{\psi}_n}, \sigma_{\dot{\psi}_n}^2$	
Acceleration	\ddot{x}_n	$\sim N$	$\mu_{\ddot{x}_n}, \sigma_{\ddot{x}_n}^2$	\ddot{x}_n
	\ddot{y}_n		$\mu_{\ddot{y}_n}, \sigma_{\ddot{y}_n}^2$	
	\ddot{z}_n		$\mu_{\ddot{z}_n}, \sigma_{\ddot{z}_n}^2$	
	$\ddot{\phi}_n$		$\mu_{\ddot{\phi}_n}, \sigma_{\ddot{\phi}_n}^2$	$\ddot{\phi}_n$
	$\ddot{\theta}_n$		$\mu_{\ddot{\theta}_n}, \sigma_{\ddot{\theta}_n}^2$	
	$\ddot{\psi}_n$		$\mu_{\ddot{\psi}_n}, \sigma_{\ddot{\psi}_n}^2$	
Measurement	ρ_n	$\sim N(\mu_{\rho_n}, \sigma_{\rho_n}^2)$	ρ_n	(rad)

3.2.2.1 Target Position Estimate.

² Estimates about the target's position need to be populated as factors into the Factor Graph being generated for the Cooperative Localization Algorithm to function.

² $x=(East-Position, North-Position, Vehicle \text{ Orientation } w \text{ respect to the North Axis})^T$

A generic triangulation method will be used. For simplicity, the factors being collected will carry no altitude measurement and instead only a 2-D space representing the East (x), North (y) coordinates of an NED (North, East, Down) coordinate system. Their motion will be modeled using the following equations for the velocity motion model [64]

$$\begin{bmatrix} E' \\ N' \\ \psi' \end{bmatrix} = \begin{bmatrix} E \\ N \\ \psi \end{bmatrix} + \begin{bmatrix} -\frac{v_t}{\omega_t} \sin(\psi) + \frac{v_t}{\psi_t} \sin(\psi + \omega_{\psi t} \Delta t) \\ \frac{v_t}{\omega_t} \cos(\psi) - \frac{v_t}{\psi_t} \cos(\psi + \omega_{\psi t} \Delta t) \\ \omega_{\psi t} \Delta t \end{bmatrix} \quad (6)$$

The pose of the vehicles at time t is represented by $x = (E, N, \psi)$ while $x' = (E', N', \psi')^T$ corresponds to the pose of the vehicles at time $t + \Delta t$. The input into the system is $u_t = (v_t, \omega_{\psi t})^T$ where v_t and $\omega_{\psi t}$ are the vehicle velocity and angular velocity at time t .

Given that each team vehicle is outfitted with camera, GPS, and odometry sensors we can model the initial estimate for the Target Vehicle's position using the following model.

$$pose_{uav1} = \begin{bmatrix} E_1, N_1, \psi_1 \end{bmatrix}^T \quad pose_{ugv1} = \begin{bmatrix} E_2, N_2, \psi_2 \end{bmatrix}^T \quad pose_{ugv2} = \begin{bmatrix} E_3, N_3, \psi_3 \end{bmatrix}^T \quad (7)$$

$$\begin{aligned} E_1 + R_1 * \sin(\rho_1) & & N_1 + R_1 * \cos(\rho_1) \\ E_T = E_2 + R_2 * \sin(\rho_2) & & N_T = N_2 + R_2 * \cos(\rho_2) \\ E_3 + R_3 * \sin(\rho_3) & & N_3 + R_3 * \cos(\rho_3) \end{aligned} \quad (8)$$

³ Setting this problem up as a system of linear equations

$$b = Ax \quad (9)$$

³ Recall ρ_n = bearing measurement to target vehicle with respect to vehicle n

$$b = \begin{bmatrix} E_1, E_2, E_3, N_1, N_2, N_3 \end{bmatrix}^T$$

$$A = \begin{bmatrix} 0 & 1 & -\sin(\rho_1) & 0 & 0 \\ 0 & 1 & 0 & -\sin(\rho_2) & 0 \\ 0 & 1 & 0 & 0 & -\sin(\rho_2) \\ 1 & 0 & -\cos(\rho_1) & 0 & 0 \\ 1 & 0 & 0 & -\cos(\rho_2) & 0 \\ 1 & 0 & 0 & 0 & -\cos(\rho_3) \end{bmatrix}$$

$$x = \begin{bmatrix} E_T, N_T, R_1, R_2, R_3 \end{bmatrix}^T$$

Then rearranging (9) into

$$x = A \backslash b \tag{10}$$

yields an estimate for the target's position (E_T, N_T) in addition to distances (R_1, R_2, R_3) between the target and each vehicle node (10).

3.3 Limitations

Due to the requirements of the Cooperative Localization algorithm, some estimates need to be included as factors within the built graph. The model in the previous section serves as the chosen method to obtain an estimate on the target's location. In order for a solution of an estimate to exist, there must be at least three vehicles with visibility on the target. Additionally, the target must also exist within the polygon footprint of the team members. Without these criteria being met, a proper estimate cannot be generated resulting in a missing factor within the graph. This missing factor implicates the optimization algorithm's ability to produce a viable

position fix on the target.

3.4 Summary

Through the use of Object Oriented design patterns, the components listed in Section 3.1 can be designed in such a way where the class methods meet the responsibility requirements within the framework. Additionally, defining class properties relevant to the information used within each component allows for both component interoperability and user interface. Robustness is highly desired within each component therefore a rapid development environment with a large support base such as MATLAB proves useful. Through the use of this IDE the testing of each component in an isolated environment becomes easier to accomplish. Lastly, the use of LCM allows for packet analysis through the *lcm-spy* inspection utility. These features used/implemented allow for better observation of the framework's performance during the joint task of Cooperative Localization in this mission set.

IV. Results

This chapter presents qualitative analysis on the design approach. Instead of performing a comparison of custom benchmarks between this architecture and an alternate, performance of individual components are discussed; demonstrating functionality of the intended design. This analysis addresses developed methods within each component and how they satisfy the component's objective within the mission and the overall framework. Additionally, the constraints and oversight behind the design and how they impact desired robustness are commented on. Section 4.1 discusses the issues encountered with the communication framework used and how they implicated the original experiment in regard to the simulation design. Section 4.2 presents the Deliberator's performance under its assigned tasks within the scope of the mission set. Analysis of this component focuses heavily on the ability to communicate with the Sequencer and Coordinator during changes in mission phases. The Sequencer component is discussed in Section 4.3. Highlighted in this discussion are the roles of the internal layers and how they satisfy the responsibilities of the component within the framework. In Section 4.4, the Coordinator performance with respect to coalition formation and joint semiosis is presented. Section 4.5 discusses the Controller component. This section focuses primarily on the Unified Behavior Framework. Described here are action generation relative to the mission set. Lastly, Section 4.7 presents the performance of the pose graph localization algorithm. Additionally, simulation properties and model assumptions are presented in tables within this section.

4.1 Component Communication

The components making up the framework consist of a number of classes, some of which operate at their own frequency. To achieve a simulation for this experiment on

one computer, a parallel process needs to run for each component in the framework of each vehicle (three total). LCM 2.2.3 would allow for these processes to run individually, while providing a communication layer between each component. Unfortunately, issues were encountered with the compatibility of LCM and MATLAB.

To transport data through LCM, a message containing the data is compartmentalized and published to a particular topic name. Any process that wishes to receive this information must know both the topic name and the content/organizational structure of the message to be able receive it properly. Then the subscriber must *subscribe* to the topic name and receive-push the messages to an aggregator. This aggregator holds a queue of these messages and with the proper commands can pop out individual messages in sequential order. In MATLAB there appears to be an issue between the publishing and receiving of messages into the aggregator. Problems were encountered when multiple messages with different topic names were published simultaneously prior to their aggregating on the subscriber end. The aggregator would receive empty messages into its queue even if the *lcm-spy* utility detected valid messages on the particular topic as being broadcasted. Because of this issue, the simulation for the frameworks in MATLAB needed to be designed on one file without LCM. To push information into the proper containers a sequential approach needed to be used. This constrained the operation of the framework to exist on a single loop for all components. In the future, development of the framework should be completed in a language that permits callbacks and threaded operations; allowing the design to proceed as intended.

4.2 Deliberator

Recalling Section 3.1.1, the role of the Deliberator is to perform high level reasoning tasks. To simplify the problem and constrain the scope to the principle of

peer flexibility, deliberation in this framework was limited to communicating critical information such as the operating area and desired position solutions for the *GoToXY* behavior with the network of vehicles. These parameters are decided by the current mission set and as such were deemed necessary to exist in the Deliberator component. To further streamline sharing of this information, the Deliberator component would share its parameters with a Deliberator state container for simplified referencing both within the vehicle framework and network. To generate a *Desired Position*, relevant to the *GoToXY* position, a few requirements must be met:

- Current mission phase must be under *Pursuit*
- Three team members need to have visibility on the target
- An estimate for the *Target* vehicle position must be generated

Due to time constraints and the stage of development, the simulation could never meet these conditions therefore no *Desired Position* could be generated. As a result, simulations in Section 4.6 does not include a simulation for the *GoToXY* behavior.

4.3 Sequencer

The plans from the Deliberator are used by the Sequencer to build a toolkit (set of behaviors) for the controller to use to accomplish the current mission. For this simulation, the Sequencer was solely responsible for passing a list of behavior names to the controller. This list was transported through LCM and received by the UBF for placement into an aggregator. This component was able to search through its library of behaviors, construct a list, and publish the behaviors with associated priorities (e.g. behavior priorities). This allowed the UBF to instantiate these behavior objects in a structure with defined priority votes. Within the Sequencer, the Behavior Planner is responsible for associating the current plan with the behaviors available within the

library. In this simulation, the planning consisted of pulling all behaviors available and storing them into this list with some preset priority values. No additional planning was made. Additionally, no feedback was provided to the Deliberator component for any replan requests. In future work, we aim to demonstrate the use of the state container, populated with a desired futures state, as the appropriate means of sharing plans between the Deliberator and Sequencer.

4.4 Coordinator

The responsibilities of the Coordinator within the framework encompass any information necessary for cooperative functionality between multiple team members. In this experiment, the design of the Coordinator was limited to forming a team coalition and the passing of useful data among agents associated as team members. In this implementation, a centralized approach to cooperative work was chosen, where the agent *UAV1* was defined as the central node of the vehicle network. This assignment meant that when it came to the task of cooperative localization and generation of desired states for the *GoToXY* behavior, *UAV1* was responsible for handing out assignments to the other team members. The design choice was due to *UAV1* being the only air vehicle in the network, and therefore having the best visibility over the operating area.

A team is constructed by requiring every friendly vehicle entering the operating area to broadcast a message including their vehicle ID as a string through an LCM channel within the local network. The central node then listens to this channel and updates the team list accordingly as new friendlies enter. Unfortunately due to the shortcomings of LCM compatibility with MATLAB described in Section 4.1, team construction could not be implemented through Coordinator methods but instead was allocated directly into the Coordinator. This same shortcoming applied to the writing of rel-

evant team agent information such as position, attitude, and measurements into the coordinator class. This information was shared with a Coordinator State on each vehicle to be accessed by the vehicles themselves for certain algorithms, such as collision sensing and the *PathFollow* behavior.

4.5 Controller

4.5.1 UBF.

Action generation is handled by the Unified Behavior Framework class. Existing within the Unified Behavior Framework are instances of other classes such as the Arbiter, StateContainer, Behaviors, and Actions. The UBF receives a list of Behaviors from the Sequencer through an LCM channel and instantiates the said behaviors in an array of objects. Using information from each behavior in this array, the Unified Behavior Framework class handles the construction of a State Container to be used exclusively in this class. The establishing of this container streamlines the inputs for arbitration during action generation. Normally, this construction would occur through the use of callbacks in LCM but, due to the compatibility issue the container was constructed and updated manually. Arbiter construction was also added as a method within the framework allowing for the instantiation of different arbiter types for use as the primary arbitration technique during the action generation. Generating an action consisted of passing in the constructed State Container into the arbiter for action generation and selection.

4.5.2 State.

The most novel contribution of this work is the concept of associating particular states within the behaviors themselves. This design practiced polymorphism for categorizing state information. Behaviors and other major components within the

framework benefit from the compartmentilization of state information into practical categories in the more streamlined design. Establishing the relationship between behaviors and states allows for a more practical approach in design/implementation of new behaviors and perceptor techniques. New behavior design is facilitated by endowing each with knowledge of required states and what categories to search for when establishing objects and applying methods. This design approach allowed the behaviors in the behavior list to report to the Unified Behavior Framework class their necessary containers for action generation. The UBF class then used this information for container construction. This same relationship can and was extended to the other layers within the framework for more practical referencing. Examples of this are the Deliberator (4.2), Coordinator (4.4), and Sequencer (4.3) states mentioned above.

4.5.3 Hardware Abstraction.

A method for interpreting the output of the arbiter and moving the vehicles was needed. To emulate an autopilot, we decided on waypoints/desired positions as action outputs. This hardware abstraction layer takes the output of the Arbiter/UBF, pulls from the necessary state containers, and uses Proportional-Derivative control to generate speed and angular velocity outputs to be used in a velocity motion model. The vehicle was then actuated and the output states were written to the respective state containers.

4.6 Behavior Simulation

To demonstrate the framework’s ability to generate actions under changing environmental conditions a few simulations are presented below. Like previously mentioned in Section 4.2, due to time constraints a *mDesiredPosition* from the Deliberator class could not be generated. This property is a requirement for the cooperative lo-

calization algorithm and as such an input for the *GoToXY* behavior. As a result, the only simulations presented are that of the *FollowPath* (4.6.1), *AvoidOthers* (4.6.2), and *FollowObject* (4.6.3) behaviors. In each figure corresponding to the simulations there exists a few vehicle attributes:

- Each vehicle and its attributes are identifiable by a constant color.
- The heading and field of view for each vehicle is depicted through two segments. The smaller angle between each segment represents this field of view which is bisected by the vehicles current heading.
- The area of interest for each vehicle is defined as the enclosed area within a polygon. The vertices of this polygon are depicted as diamonds.
- A collision bubble corresponding to each vehicle is identifiable through a dotted circle centered on the corresponding vehicle's position.

4.6.1 FollowPath.

The purpose of this simulation is to demonstrate the *FollowPath* behavior's ability to accomplish two things:

- Generate an area of interest for a vehicle within a defined Operating Area
- Generate a set of waypoints within this area of interest for patrolling

Table 3 presents the definitions made to each class for this simulation. Because we want to illustrate the vehicle's ability to define an area of interest and remain inside it for patrolling purposes, the priority values for every other behavior are set as 0. This allows for the *WinnerTakeAll* arbiter to always select the action generated by the *FollowPath* behavior.

Table 3. FollowPath Simulation Definitions

Behavior Class	Property	Value
AvoidOthers	<i>mPriority</i>	0
GoToXY	<i>mPriority</i>	0
FollowObject	<i>mPriority</i>	0
FollowPath	<i>mPriority</i>	0.25

Figure 8 illustrates the environmental conditions at the beginning of this simulation. The area of interest for each vehicle is generated through the *FollowPath* behavior by defining a cluster center and then dividing the Operating Area relative to this cluster center. The objective for the vehicle beyond this point is to initiate a patrol sequence in which its operating area is scanned for any signs of vehicles/objects not identified as team members. Due to the class definitions assigned for this particular simulation, the vehicle will continue to patrol even when a target of interest is identified by its perceptor. In Figure 9 we can see that even after the Target has exited the Operating Area, the vehicles continue to patrol their assigned sector.

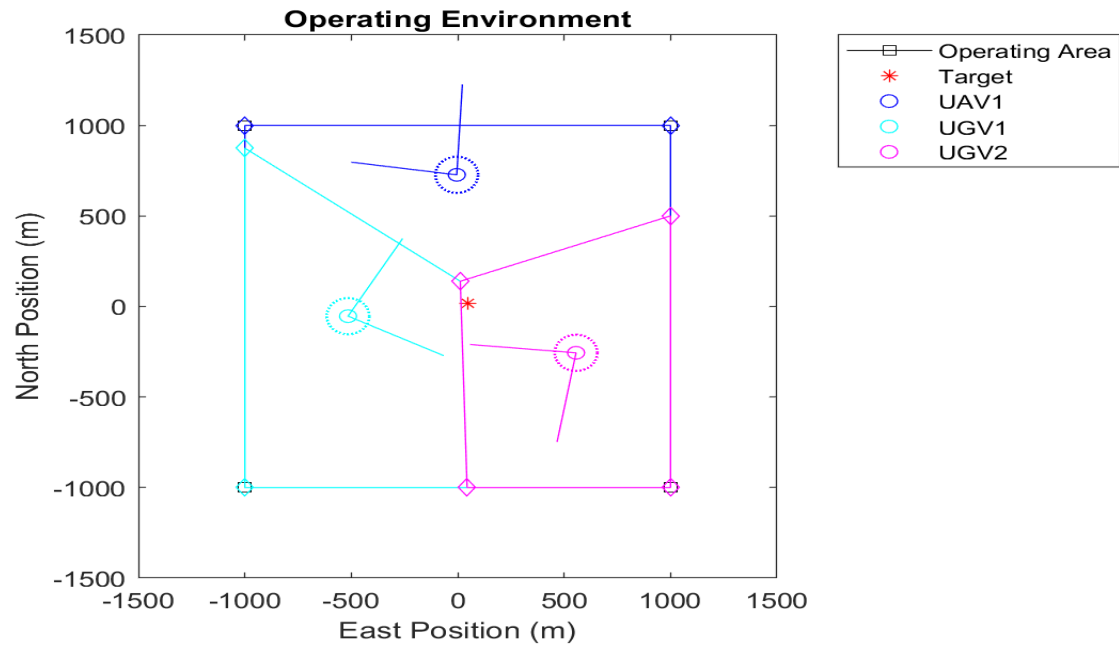


Figure 8. Starting conditions for *FollowPath* behavior simulation.

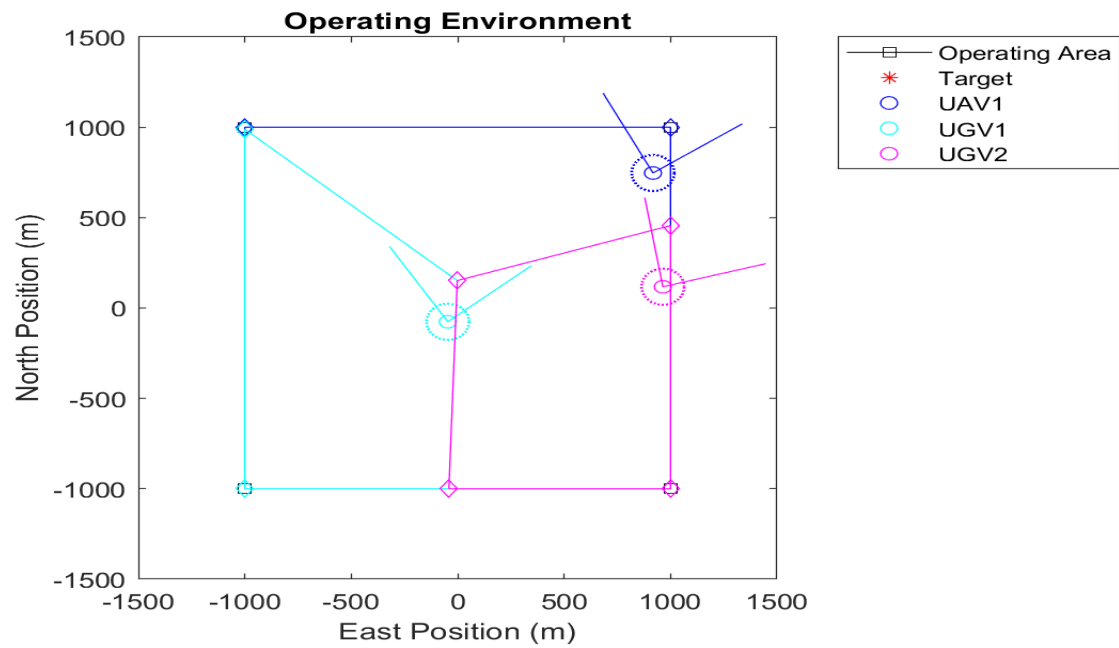


Figure 9. End of simulation for *FollowPath* behavior. Vehicles continue to patrol in their assigned sectors.

4.6.2 AvoidOthers.

The purpose of this simulation is to demonstrate the *AvoidOthers* behavior's ability to evade a collision scenario for a vehicle. To accomplish this, the *AvoidOthers* behavior requires a collision bubble radius which is defined and then inherited by a collision prevention perceptor. If an object exists inside this defined radius then the priority value for this behavior becomes 1. This prompts the *WinnerTakeAll* arbiter to select the action generated by the *AvoidOthers* behavior. The class definitions required for this demonstration are presented in Table 4.

Table 4. AvoidOthers Simulation Definitions

Behavior Class	Property	Value
AvoidOthers	<i>mPriority</i>	[0, 1]
	<i>mBubbleRadius</i>	100m
GoToXY	<i>mPriority</i>	0
FollowObject	<i>mPriority</i>	0
FollowPath	<i>mPriority</i>	0.25

Illustrated in Figure 10 are the initial environmental conditions for the simulation of this behavior. From the start we can see that an object/obstacle (Target) exists within the defined collision bubble of *UGV2*. According to its *FollowPath* behavior, the vehicle should initially head to the top right corner of its assigned sector but will choose not to because of an object existing within its collision radius. Additionally, due to the class definitions for this experiment the vehicle will also avoid pursuit. Its primary objective in this scenario will be to avoid collision and then resume patrol of its sector.

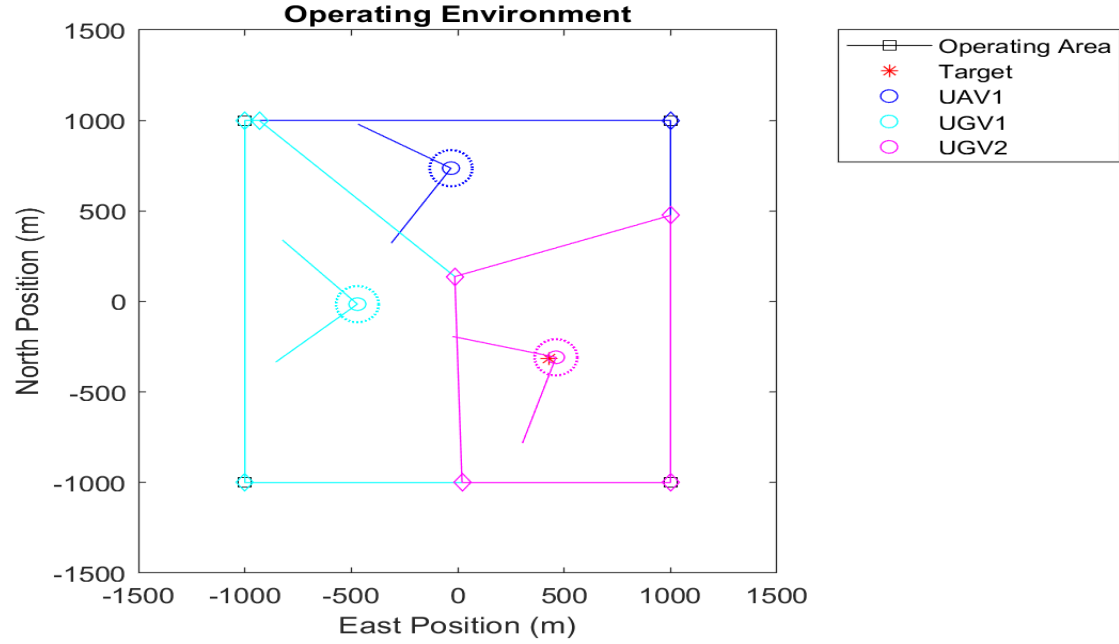


Figure 10. Starting conditions for *AvoidOthers* behavior simulation. Object exists inside collision bubble radius. *UGV2*'s first waypoint for *FollowPath* behavior is top right corner.

Depicted in Figure 11 is *UGV2*'s reaction to the detection of an object inside its collision radius. It abandons the *FollowPath* behavior's waypoint and adjusts its heading to drive itself in a direction to avoid collision. Upon successful evasion (Figure 12), it then reverts to its *FollowPath* behavior (Figure 13).

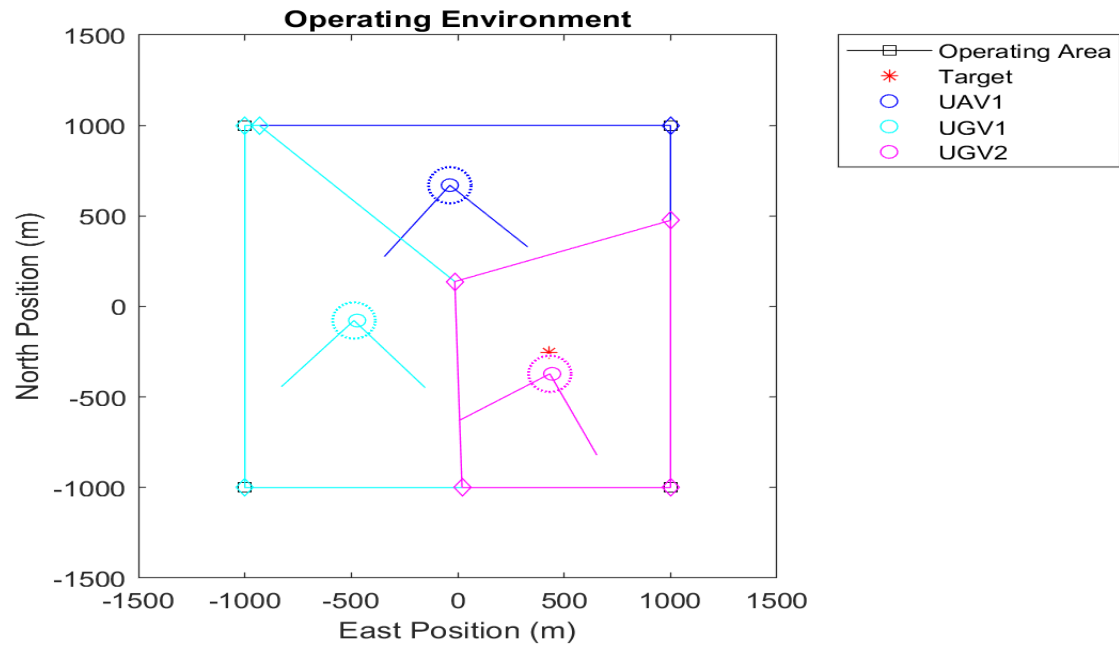


Figure 11. Reaction of *AvoidOthers* behavior when object exists inside collision bubble.

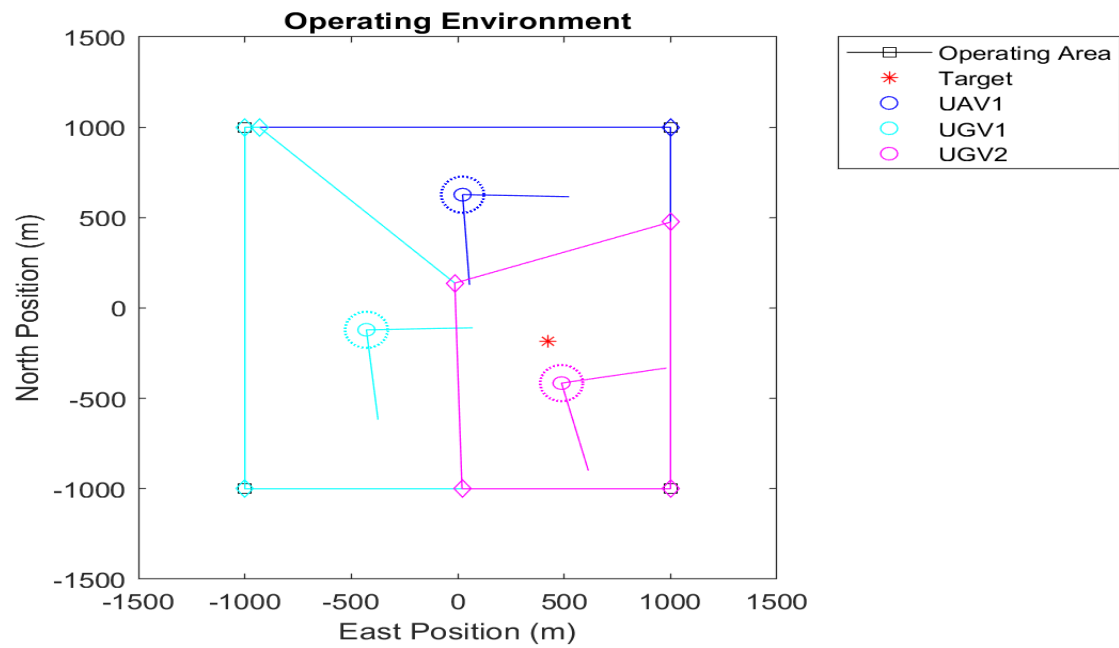


Figure 12. *UGV2* successfully evades collision.

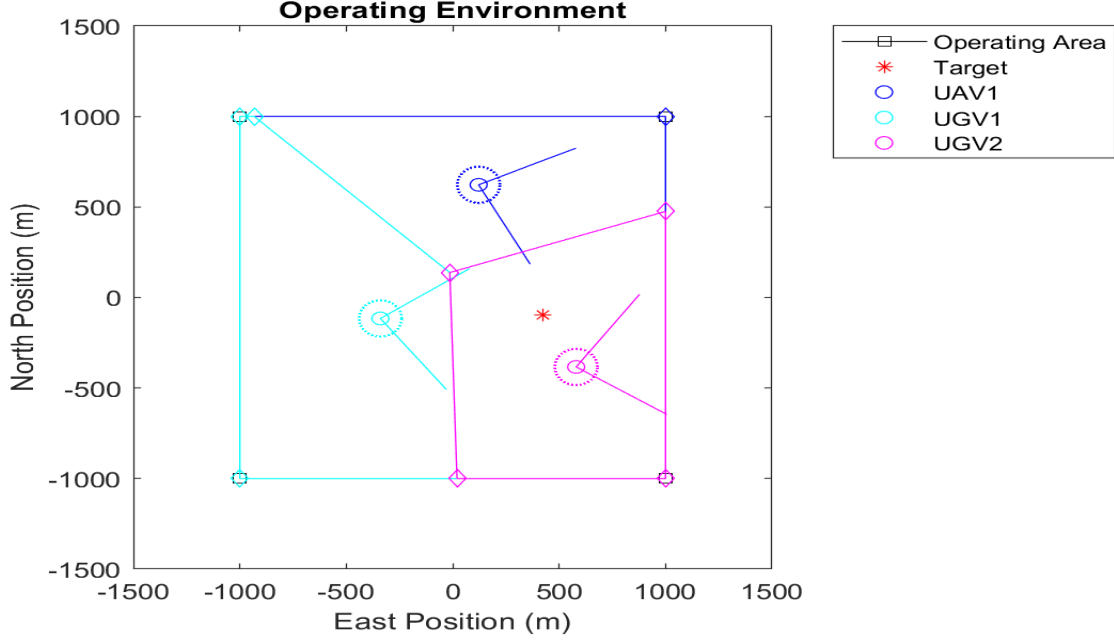


Figure 13. *UGV2* resumes *FollowPath* behavior after collision evasion.

4.6.3 FollowObject.

When simulating the *FollowObject* behavior it is important to demonstrate its ability to abandon all behaviors and engage in a pursuit of the Target vehicle. This is necessary for the eventual localization phase of the defined mission. To do this Table 5 presents the class definitions necessary for this experiment. In the event that an unidentified object enters the field of view a team vehicle, the *FollowObject* behavior's action priority supersedes all other defined behavior actions. This condition is enabled by the reception of a camera perceptor's bearing measurement. This measurement is provided as an input to the *FollowPath* behavior which is then used in conjunction with its *mDesiredRange* property to generate a waypoint in the direction of said bearing.

Table 5. FollowObject Simulation Definitions

Behavior Class	Property	Value
AvoidOthers	<i>mPriority</i>	$[0, 1]$
	<i>mBubbleRadius</i>	$100m$
GoToXY	<i>mPriority</i>	0
FollowObject	<i>mPriority</i>	$[0, 0.75]$
	<i>mDesiredRange</i>	$500m$
FollowPath	<i>mPriority</i>	0.25

Illustrated in Figure 14 is a scenario in which a Target exists in the defined Operating Area. Initially, *UGV1* is the only vehicle with visibility on the Target. As the simulation continues, *UGV1* eventually loses sight of the Target while *UAV1* and *UGV2* gain visibility of it. Figure 15 illustrates *UGV1* patrolling its sector while *UAV1* returns to its *FollowPath* behavior due to loss of sight on the Target and *UGV2* continues its pursuit outside of the Operating Area. Under ideal simulation conditions, the *GoToXY* behavior would then engage prompting all vehicles to continue pursuit cooperatively. Unfortunately, due to time constraints this is not demonstrated.

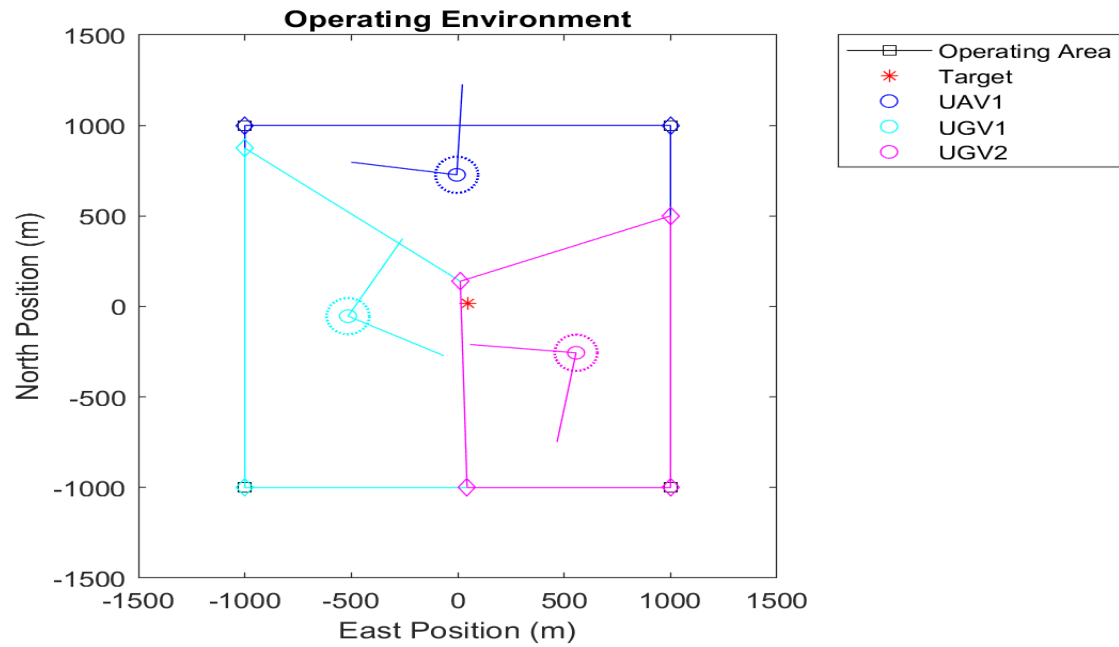


Figure 14. Starting conditions for *FollowObject* behavior simulation.

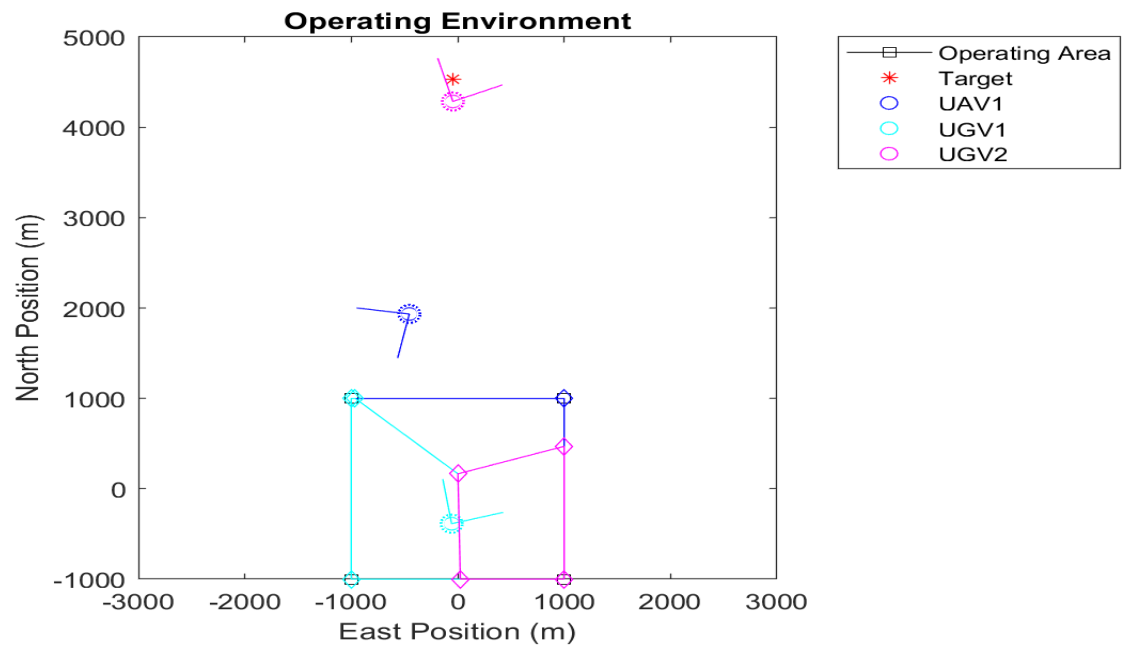


Figure 15. *UAV1* has lost sight of Target and resumes *FollowPath* behavior. *UGV2* continues to pursue *FollowObject* behavior.

4.7 Cooperative Localization

The Cooperative Localization algorithm exists as a class to be used within the coordinator/coordinatorState container. The solution, like mentioned above is provided to an algorithm that is then used to write a desired position to the Deliberator and the appropriate state container. Due to time constraints, the class could not be implemented. However some plots below represent solutions from the proposed approach, only using a batch method called the Levenberg–Marquardt optimization technique. The figures demonstrate that the proposed approach is valid and can provide a solution, yet the computation complexity grows quickly with the graph size and would not be valid for real-time scenarios. Future work is to implement the same approach with the iSAM2 library, which only optimizes over the most recent portions of the graph, allowing real-time computations.

4.7.1 Control Variables.

Table 6 contains the variables that will control the vehicles to simulate the system over time. Both Target Vehicle and Team vehicles will follow the velocity motion (6) with each input being a random variable with a uniform distribution within an certain range.

- v_i denotes the input speed of the vehicle in **m/s**
- ω_i denotes the input angular velocity in **rad/sec**

4.7.2 Response Variables.

Estimating the location of a target vehicle using this *pose graph localization* approach requires initial estimates of the target’s pose for the algorithms optimization method. To solve for these estimates, the Response Variables listed in Table 7 will be

Table 6. Control Variables

Variable/Factor	Normal Operating Level and Range	Proposed Setting	Relationship to Response Variables (Predicted Effect)
v_{targ}	10 : 25(m/s)	$v_{targ} \sim N(20, 5)$	(6)
ω_{targ}	$-\frac{\pi}{2} : \frac{\pi}{2}(rad/s)$	$\omega_{targ} \sim N(0, \frac{\pi}{2})$	(6)
v_{uav1}	10 : 25(m/s)	$v_{uav1} \sim N(20, 5)$	(6)
ω_{uav1}	$-\frac{\pi}{2} : \frac{\pi}{2}(rad/s)$	$\omega_{uav1} \sim N(0, \frac{\pi}{2})$	(6)
v_{ugv1}	10 : 25(m/s)	$v_{ugv1} \sim N(20, 5)$	(6)
ω_{ugv1}	$-\frac{\pi}{2} : \frac{\pi}{2}(rad/s)$	$\omega_{ugv1} \sim N(0, \frac{\pi}{2})$	(6)
v_{ugv2}	10 : 25(m/s)	$v_{ugv2} \sim N(20, 5)$	(6)
ω_{ugv2}	$-\frac{\pi}{2} : \frac{\pi}{2}(rad/s)$	$\omega_{ugv2} \sim N(0, \frac{\pi}{2})$	(6)

used in conjunction with (9). This equation requires that a minimum of two nodes must have visibility on the target at all times. Therefore as stated in Section 3.2.2, we will assume all three vehicles have *lag-less* communication and constant visibility on the target to simplify the simulated environment during the *Pursuit* phase.

- **Team Vehicle Pose** is denoted as (E_n, N_n, ψ_n) , with E_n and N_n corresponding to the East and North coordinate the vehicle in the NED-Frame. ψ_n represents the vehicle's Northern bearing with respect to the North Axis.
- **Target Vehicle Location** is denoted as (E_{targ}, N_{targ}) with E_{targ} and N_{targ} corresponding to the East and North coordinate the vehicle in the NED-Frame.
- **Bearing to Target Vehicle** with respect to the Vehicle **n**'s Northern Axis is denoted by ρ_n

Table 7. Response Variables

Response Variable	Normal Operating Level and Range	Measurement Precision and Accuracy	Relationship to Objective
E_{uav1}	∞	$E_1 \sim N(\mu_{N1}, \sigma_{N1}^2)$	$E_{targ} = E_1 + R_1 \sin(\rho_1)$
N_{uav1}	∞	$N_1 \sim N(\mu_{E1}, \sigma_{E1}^2)$	$N_{targ} = N_1 + R_1 \sin(\rho_1)$
ψ_{uav1}	$0 - 2\pi$	$\psi_1 \sim N(\mu_{\psi_1}, \sigma_{\psi_1}^2)$	$\rho_1 = \text{atan2}(N_{targ} - N_1, E_{targ} - E_1) - \psi_1$
E_{ugv1}	∞	$E_2 \sim N(\mu_{N2}, \sigma_{N2}^2)$	$E_{targ} = E_2 + R_2 \sin(\rho_2)$
N_{ugv1}	∞	$N_2 \sim N(\mu_{E2}, \sigma_{E2}^2)$	$N_{targ} = N_2 + R_2 \sin(\rho_2)$
ψ_{ugv1}	$0 - 2\pi$	$\psi_2 \sim N(\mu_{\psi_2}, \sigma_{\psi_2}^2)$	$\rho_2 = \text{atan2}(N_{targ} - N_2, E_{targ} - E_2) - \psi_1$
E_{ugv2}	∞	$E_3 \sim N(\mu_{N3}, \sigma_{N3}^2)$	$E_{targ} = E_3 + R_3 \sin(\rho_3)$
N_{ugv2}	∞	$N_3 \sim N(\mu_{E3}, \sigma_{E3}^2)$	$N_{targ} = N_3 + R_3 \sin(\rho_3)$
ψ_{ugv2}	$0 - 2\pi$	$\psi_3 \sim N(\mu_{\psi_3}, \sigma_{\psi_3}^2)$	$\rho_3 = \text{atan2}(N_{targ} - N_3, E_{targ} - E_3) - \psi_1$

4.7.3 Constant Factors.

In this experiment there are a few factors that must remain constant in order to keep data integrity. These factors are also help constrain the scope of what is being analyzed. Table 8 summarizes these factors.

Table 8. Constant Factors

Variable/Factor	Desired Experimental Level	Measurement Precision	Relationship to Response Variables (Predicted Effect)
Vehicles in Operating Space	3	N/A	(10)
Mobility Model	Velocity Motion	$\sigma_v = 0, \sigma_\omega = 0$	(6)
GPS Signal Integrity	N-E-D Local Level Coordinate Frame	<i>Table 7</i>	<i>Table 7</i>
Camera Quality	85° FOV	$\sigma_v = 1^\circ, \sigma_\omega = 1^\circ$	$\psi_n = \text{atan2}(N_{targ} - N_n, E_{targ} - E_n) - \rho$
Transmit Latency	~ 0 sec	N/A	Needed ~ 0 possible to constrain solution for target position

Figure 16 depicts the computation time for the algorithm, and shows easily why the iSAM2 approach is a more desirable due to its factorization of the local factors instead of the whole graph. This would result in a quicker computation time and a much flatter curve.

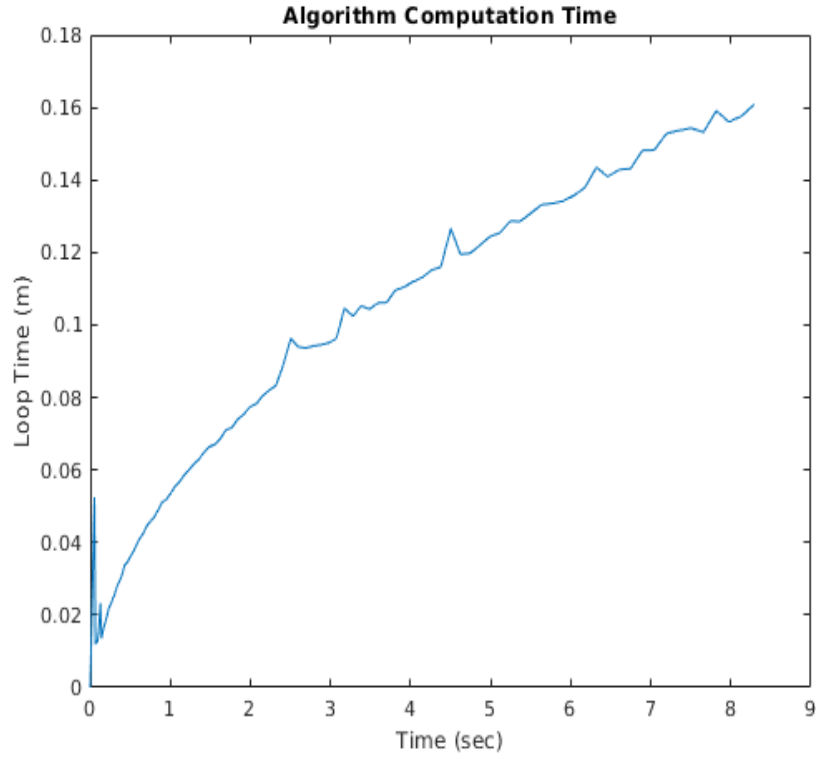


Figure 16. Computation time for pose graph Cooperative Localization algorithm over time using Levenberg–Marquardt optimizer.

Figure 17, demonstrates that the approach is valid and can produce an estimate of the target position, given sufficient number of bearings from the team members. Please note that the noise was removed from the system for validation purposes. There was not time available to add the uncertainty back in.

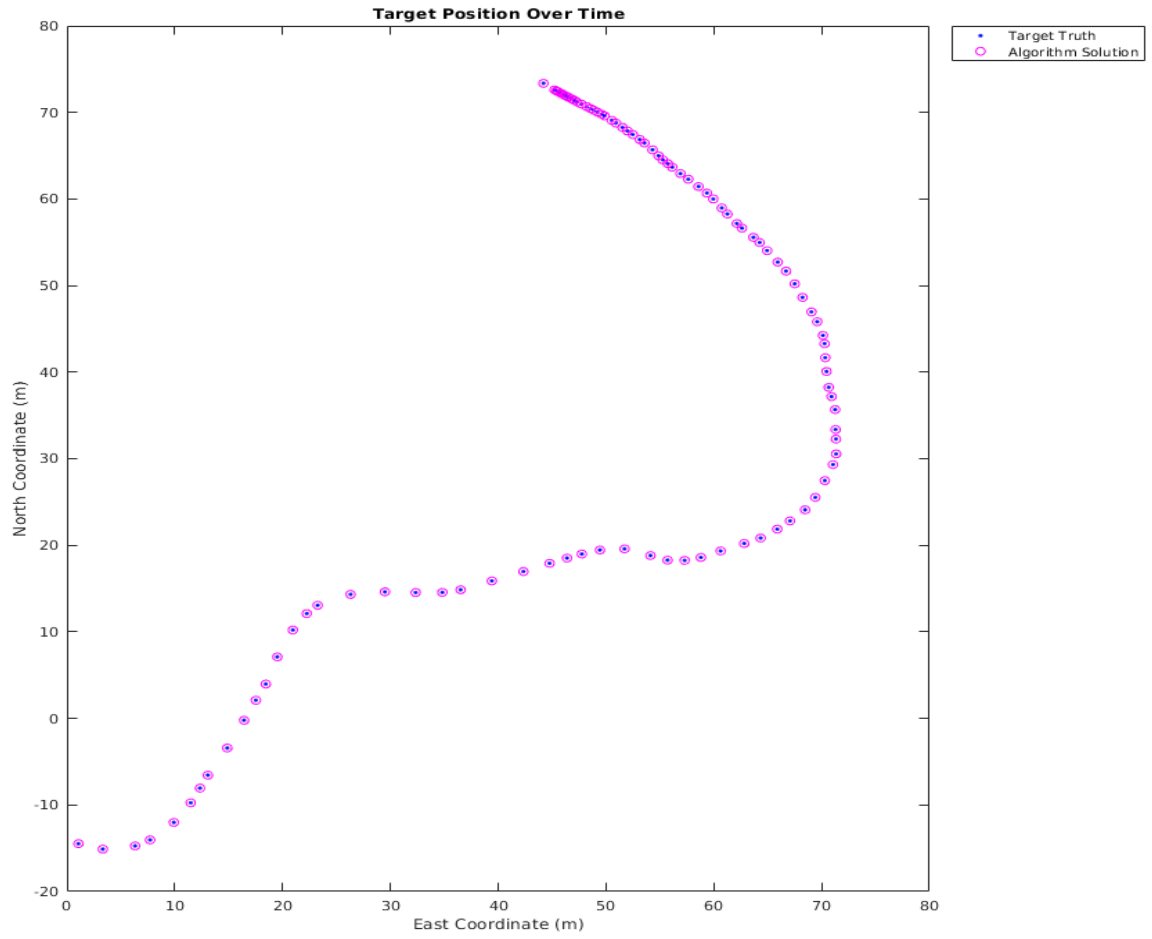


Figure 17. Noise-less Position of Target over time with applied Cooperative Localization algorithm; Levenberg–Marquardt Optimizer.

V. Conclusion

The research provided in this document presents the development towards a testing platform for autonomy. The framework developed uses the object oriented programming environment available through MATLAB due to its similarities with Java[™] and C++. Additionally, MATLAB's documentation and debugging tools facilitated the development process. To test its functionality and evaluate the principle of *peer flexibility* in autonomy the framework was instantiated on three vehicles tasked with the responsibility of working cooperatively to patrol a predefined Operating Area and localize any non-friendly vehicles that enter. To accomplish this each vehicle is outfitted with perceptors that aid in obtaining bearing information on vehicles in their line of sight and collision prevention. The information obtained through these perceptors are used in conjunction with GPS and INS data to enable their behavior based control.

5.1 Summary

Due to constraints with development time not all behaviors were fully tested therefore a localization phase for this experiment could never be attempted. Fortunately, a general algorithm for a pose graph cooperative localization approach was tested and demonstrated using the GTSAM 3.2.1 library. The Levenberg–Marquardt optimization technique used in this localization test revealed that a quicker algorithm such as the iSAM2 is desired for practical performance. Although tests for the Unified Behavior Framework and a localization algorithm were demonstrated independently, a few deliberate oversights needed to be made in order to reach this stage of development, which are discussed in Chapter III.

5.2 Contribution

The state component of this framework is essential for effective framework operation. By design, the state component stores the dynamic information critical for decision making within the platform. Previous work on HAMR and the UBF used a World Model approach where any time a component needed information about its environment it would be passed in a reference to this model. In order to establish a pattern for design we found that compartmentalization of this World Model facilitated behavior design. If during the design process for a behavior it was given knowledge of the states required and how to find them, then modularity in behavior implementation across different vehicle platforms could be more easily achieved. This relation could also be extended to the different layers within the framework which use and store unique state information. In this work, the compartmentalization of this World Model consists of the following state containers: *PositionState*, *AttitudeState*, *VelocityState*, *AccelerationState*, *PerceptorState*, *DeliberatorState*, *SequencerState*, and a *CoordinatorState*. Each container consists of information relevant to its name so that any time a component or a function needs to access specific information it looks to the parent container for availability. If the information is present then it can be pulled, if it is not then it simplifies the process of determining what information within that category is available. This organizational structure facilitates access to information that can be represented in different ways such as position data, which can be expressed in different coordinate frames. Additionally, within the controller layer the UBF could construct a container consisting of state information relevant **only** to the behaviors within its *behaviorList* as opposed to passing in a reference to the entire World Model.

5.3 Final Remarks

Ultimately, this work presents some developmental insight on the shortcomings of previous implementations for this framework. It addresses some of the obstacles encountered in the early stages of development for this platform and provides a general starting point for a robust end product. Fundamentally, the design of the framework shown in Figure 6 exhibits properties of the desired testing platform discussed in Section 1.3. Although MATLAB proves to be a useful environment for the conceptual implementation of a single framework, simulation of multiple frameworks working cooperatively ideally should be done outside of this chosen design environment. Further refinement of each component pertaining to each layer of the framework is needed to address the deliberate oversights made in this work.

Bibliography

1. “America’s Air Force: A Call to the Future,” tech. rep., United States Air Force, 2014.
2. Office of the Chief Scientist, “Autonomous Horizons: System Autonomy in the Air Force - A Path to the Future,” tech. rep., United States Air Force, 2015.
3. J. Peterson, Gilbert L., Leishman, Robert C., “AFIT White Paper: Autonomous and Cooperative System Framework and Reference Implementation.”.
4. E. Tsardoulis and P. Mitkas, “Robotic Frameworks, Architectures and Middleware Comparison.” arXiv:1711.06842, [cs.RO], 2017.
5. E. Gat, “Artificial intelligence and mobile robots,” ch. On Three-layer Architectures, pp. 195–210, Cambridge, MA: MIT Press, 1998.
6. R. R. Murphy, *AI Robotics*. Cambridge, MA: The MIT Press, 2000.
7. N. J. Nilsson and D. L. Nielson, “Shakey the Robot,” Tech. Rep. 323, Stanford University, 1984.
8. D. M. Roberson, “The unified behavior framework for the simulation of autonomous agents,” Master’s thesis, Air Force Institute of Technology, 3 2015.
9. T. B. Bodin, “A Task-Flexible Platform for Autonomous Unmanned Aerial Systems,” Master’s thesis, Air Force Institute of Technology, 3 2018.
10. R. C. Arkin, “Survivable robotic systems: Reactive and homeostatic control,” in *Robotics and Remote Systems for Hazardous Environments* (M. Jamshidi and P. Eicker, eds.), pp. 135–154, Prentice-Hall, 1993.

11. A. J. Kamrud, "Unified Behavior Framework for Discrete Event Simulation Systems," Master's thesis, Air Force Institute of Technology, 3 2015.
12. V. Braitenberg, *Vehicles: Experiments in Synthetic Psychology*. Cambridge, MA: MIT Press, 1984.
13. R. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 14–23, 1986.
14. J. P. Duffy, "Dynamic Behavior Sequencing in a Hybrid Robot," Master's thesis, Air Force Institute of Technology, 3 2008.
15. S. S. Lin, "Unified Behavior Framework in an Embedded Robot Controller," Master's thesis, Air Force Institute of Technology, 3 2009.
16. Fitz-Gibbon, T.R., "Subsumption architecture." Friedrich-Alexander-Universität EEL 6838 University Lecture, 2004. [Online; accessed December 17, 2018].
17. J. H. Connell, "A Behavior-Based Arm Controller," Tech. Rep. 1025, Massachusetts Institute of Technology, Cambridge, MA, 1988.
18. R. A. Schmidt, "A Schema Theory of Discrete Motor Skill Learning," *Psychological Review*, 1975.
19. R. C. Arkin, "Motor Schema - Based Mobile Robot Navigation," *International Journal of Robotics Research*, vol. 8, no. 4, pp. 92–112, 1989.
20. B. G. Woolley, "Unified Behavior Framework for Reactive Robot Control in Real-Time Systems," Master's thesis, Air Force Institute of Technology, 3 2007.
21. L. P. Kaelbling, "An Architecture for Intelligent Reactive Systems," Tech. Rep. 400, Stanford University, 1986.

22. J. K. Rosenblatt, “DAMN: A Distributed Architecture for Mobile Navigation.” 1997.
23. J. K. Rosenblatt, “Utility Fusion: Map-Based Planning in a Behavior-Based System,” tech. rep., University of Sydney, 1998.
24. S. Ceriani and M. Migliavacca, “Middleware in Robotics,” tech. rep., Internal Report for Advanced Methods of Information Technology for Autonomous Robotics, 2012.
25. M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” vol. 3, 01 2009.
26. Google, “Google protocol-buffers.” <https://developers.google.com/protocol-buffers>, 2018.
27. A. S. Huang, E. Olson, and D. C. Moore, “Lcm: Lightweight communications and marshalling,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4057–4062, Oct 2010.
28. I. A. D. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, and I.-H. Shu, “Claraty: Challenges and steps toward reusable robotic software,” *International Journal of Advanced Robotic Systems*, vol. 3, 03 2006.
29. I. A. D. Nesnas, A. Wright, M. Bajracharya, R. Simmons, and T. Estlin, “Claraty and challenges of developing interoperable robotic software,” in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No. 03CH37453)*, vol. 3, pp. 2428–2435 vol.3, Oct 2003.
30. B. S. E. K. Nesnas, Issa A, Wright, Anne, “Claraty: an architecture for reusable robotic software,” vol. 5083, pp. 5083 – 5083 – 12, 2003.

31. I. A. D. Nesnas, *The CLARAty Project: Coping with Hardware and Software Heterogeneity*, pp. 31–70. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.
32. R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, “The claraty architecture for robotic autonomy,” in *2001 IEEE Aerospace Conference Proceedings*, vol. 1, pp. 121–132, March 2001.
33. C. Jang and S.-I. Lee, “Opros: A new component-based robot software platform,” *ETRI Journal*, vol. 32, pp. 646–656, 10 2010.
34. B. Song, S. Jung, C. Jang, and S. Kim, “An introduction to robot component model for opros (open platform for robotic services),” *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pp. 529–603, SIMPAR, 2008.
35. J. Kim, H. Yoon, S. Kim, and S. H. Son, “Fault management of robot software components based on opros,” in *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pp. 253–260, March 2011.
36. K. Lim, S. C. Ahn, Y. Kwon, and K. Sohn, “Upnp single event mechanism for opros robot s/w platform,” in *ICCAS 2010*, pp. 981–983, Oct 2010.
37. H. Bruyninckx, “Open robot control software: the orocos project,” in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 3, pp. 2523–2528 vol.3, May 2001.
38. T. H. J. Collett, B. Macdonald, and B. Gerkey, “Player 2.0: Toward a practical robot programming framework,” *Proceedings of the 2005 Australasian Conference on Robotics and Automation, ACRA 2005*, 08 2008.

39. B. P. Gerkey, R. Vaughan, and A. Howard, “The player/stage project: Tools for multi-robot and distributed sensor systems,” *Proceedings of the International Conference on Advanced Robotics*, 08 2003.
40. D. S. Michal and L. Etzkorn, “A comparison of player/stage/gazebo and microsoft robotics developer studio,” in *Proceedings of the 49th Annual Southeast Regional Conference*, ACM-SE ’11, (New York, NY, USA), pp. 60–66, ACM, 2011.
41. G. Metta, P. Fitzpatrick, and L. Natale, “Yarp: Yet another robot platform,” *International Journal on Advanced Robotics Systems*, 2006.
42. S. Stefansson, B. Jónsson, and K. Thórisson, “A yarp-based architectural framework for robotic vision applications,” *VISAPP*, pp. 65–68, 2009.
43. R. W. Beard, T. W. McLain, D. B. Nelson, D. Kingston, and D. Johanson, “Decentralized Cooperative Aerial Surveillance Using Fixed-Wing Miniature UAVs,” *Proceedings of the IEEE*, 2006.
44. A. Chakraborty, C. N. Taylor, R. Sharma, and K. M. Brink, “Cooperative Localization for Fixed Wing Unmanned Aerial Vehicles,” in *Proceedings of the IEEE/ION Position, Location and Navigation Symposium, PLANS 2016*, 2016.
45. V. Indelman, E. Nelson, N. Michael, and F. Dellaert, “Multi-Robot Pose Graph Localization and Data Association from Unknown Initial Relative Poses via Expectation Maximization,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2014.
46. H. Li and F. Nashashibi, “Multi-Vehicle Cooperative Localization Using Indirect Vehicle-to-Vehicle Relative Pose Estimation,” in *International Conference on Vehicular Electronics and Safety*, (Istanbul, Turkey), IEEE, 2012.

47. R. Sharma and D. Pack, "Cooperative Sensor Resource Management to Aid Multi Target Geolocalization Using a Team of Small Fixed-wing Unmanned Aerial Vehicles," in *AIAA Guidance, Navigation, and Control (GNC) Conference*, 2013.
48. X. Shen, H. Andersen, W. K. Leong, H. X. Kong, M. H. Ang, and D. Rus, "A General Framework for Multi-Vehicle Cooperative Localization Using Pose Graph." arXiv:1704.01252, [cs.RO], 2017.
49. P. S. Maybeck, *Stochastic Models, Estimation, and Control*, vol. 1 of *Mathematics in Science and Engineering*. Orlando, FL: Academic Press Inc., 1979.
50. A. Grewal, Mohinder S., "Applications of Kalman Filtering in Aerospace 1960 to the Present," *IEEE Control Systems*, 2010.
51. R. B. Davis, "Applying Cooperative Localization to Swarm UAVs Using an Extended Kalman Filter," Master's thesis, Naval Postgraduate School, 9 2014.
52. F. Dellaert and M. Kaess, "Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing," *International Journal of Robotics Research*, 2006.
53. F. R. Kschischang, "Factor Graphs and the Sum-Product Algorithm," *IEEE Transactions on Information Theory*, 2001.
54. F. Dellaert and M. Kaess, "Factor Graphs for Robot Perception," *Foundations and Trends in Robotics*, 2017.
55. M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "ISAM2: Incremental Smoothing and Mapping Using the Bayes Tree," *International Journal of Robotics Research*, 2012.

56. D. J. Hooper, “A Hybrid Multi-Robot Control Architecture,” Master’s thesis, Air Force Institute of Technology, 3 2008.
57. A. S. Huang, E. Olson, and D. C. Moore, “LCM: Lightweight Communications and Marshalling,” in *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, 2010.
58. D. Hooper and G. Peterson, “HAMR: A Hybrid Multi-Robot Control Architecture,” in *Proceedings of the Twenty-Second International FLAIRS Conference*, 2009.
59. K. Cousin and G. L. Peterson, “A Distributed Approach To Solving Constrained Multiagent Task Scheduling Problems,” in *Proceedings of the 2007 AAAI Fall Symposium-Regarding the "Intelligence" in Distributed Intelligent Systems (RIDIS)*, 2007.
60. N. K. C. Krothapalli and A. V. Deshmukh, “Distributed Task Allocation in Multi-Agent Systems,” tech. rep., University of Massachusetts Amherst, 2003.
61. B. G. Woolley, G. L. Peteron, and J. T. Kresge, *Autonomous Robots*, vol. 30, ch. Real-Time Behavior-Based Robot Control, p. 233. 2011.
62. B. G. Woolley and G. L. Peterson, “Unified Behavior Framework for Reactive Robot Control,” *Journal of Intelligent and Robotic Systems*, 2009.
63. E. Gamma, *Design Patterns: Elements of Reusable Object-Oriented Software*. 1995.
64. S. Thrun, B. Wolframe, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: The MIT Press, 2006.

REPORT DOCUMENTATION PAGE					<i>Form Approved</i> <i>OMB No. 0704-0188</i>	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY)		2. REPORT TYPE		3. DATES COVERED (From — To)		
21-03-2019		Master's Thesis		Sept 2017 — Mar 2019		
4. TITLE AND SUBTITLE				5a. CONTRACT NUMBER		
A Multi-Vehicle Cooperative Localization Approach for an Autonomy Framework				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S)				5d. PROJECT NUMBER		
Edwin A. Mora, 2d Lt.				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)				8. PERFORMING ORGANIZATION REPORT NUMBER		
Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765				AFIT-ENG-MS-19-M-046		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)		
CERDEC U.S. ARMY RDECOM COMM (443) 395-0103 Email: john.a.miranda4.civ@mail.mil				RDECOM/CERDEC		
11. SPONSOR/MONITOR'S REPORT NUMBER(S)						
12. DISTRIBUTION / AVAILABILITY STATEMENT						
DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT						
A testing platform for the validity of autonomy approaches is desirable due to the Air Force's growing interest on the inclusion of autonomy into their systems. A system demonstrating autonomy displays evidence of the following three principles: task, peer, and cognitive flexibility. This research aims to focus primarily on the principle of peer flexibility, specifically the cooperative localization of desired targets in an operational environment using the combined sensors of a team of autonomous vehicles. An instantiation of a new, AFIT-developed autonomous framework will be created to solve the problem of centralized cooperative localization of a target. A key component of this framework is the Unified Behavioral Framework (UBF), which is a systematic approach of breaking down tasks, making assignments and enabling autonomous behaviors for vehicles.						
15. SUBJECT TERMS						
Autonomous Systems, Cooperative Localization, Factor Graphs, Hybrid Deliberative/Reactive Architectures, Multi-Robot Systems						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT		18. NUMBER OF PAGES	
a. REPORT	b. ABSTRACT	c. THIS PAGE			19a. NAME OF RESPONSIBLE PERSON Dr. Robert C. Leishman, AFIT/ENG	
U	U	U	UU		19b. TELEPHONE NUMBER (include area code) (937) 255-3636, x4755; robert.leishman@afit.edu	